I hereby certify that this correspondence is being deposited with the United States Postal Service as first class mail in a container addressed to: Mail Stop Fee Amendment, Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450 on April 21, 2004.

Signature: Crystal Ross
Crystal Ross

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant(s):	William W. Smith III, et al.	Group Art Unit:	3629
Serial No.:	09/684,010	Examiner:	Edward R. Cosimano
Filed:	October 6, 2000	Date:	April 21, 2004
Title:	Online, Multi-Carrier, Multi-Service Parcel Shipping Management Functional Alignment of Computer Devices		
Atty Dckt No.: PSTM0002/MRK			

COVER SHEET FOR DECLARATION UNDER 37 C.F.R. 1.131

Mail Stop Fee Amendment
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Commissioner:

Filed concurrently herewith under 37 C.F.R. § 1.131 is a Declaration bearing execution by the inventors of the above-identified application, submitting evidence substantiating conception of the claimed invention prior to March 23, 1999 coupled with due diligence through October 6, 1999, the date on which a provisional application, to which the present application claims priority, was filed.

Attached to the Declaration are Exhibits A through J. Exhibits A, B, and D through I are true and correct copies of excerpts from a document identified to me as, and on information and belief, I believe to be, a true and correct copy of the Project Wolverine Design Document, bearing a date prior to March 23, 1999. Exhibit C is a true and correct copy of a document identified to me as, and on information and belief, I believe to be, a true and correct copy of an Agreement between College Enterprises

Inc. ("CEI") and MoveIt! Software Inc. ("MoveIt!"). Exhibit J is a true and correct copy of a series of documents identified to me as, and on information and belief, I believe to be, a true and correct hardcopy of electronic mail messages ("emails") exchanged between the inventors and/or other members of the MoveIt!/iShip development team.

Respectfully submitted,

April 21, 2004
Date

Marilyn R. Khorsandi
Attorney/Agent for Applicant(s)
Reg. No. 45,744

Khorsandi Patent Law Group, ALC
140 S. Lake Ave., Ste. 312
Pasadena, California 91101-4710
Telephone No.: (626) 796-2856
Fascimile: (626) 796-2864

I hereby certify that this correspondence is being deposited with the United States Postal Service as first class mail in a container addressed to: Mail Stop Fee Amendment, Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450 on April 21, 2004.

Signature: Crystal Ross
Crystal Ross

PATENT

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant(s):	William W. Smith III, et al.	Group Art Unit:	3629
Serial No.:	09/684,010	Examiner:	Edward R. Cosimano
Filed:	October 6, 2000		
Title:	Online, Multi-Carrier, Multi-Service Parcel Shipping Management Functional Alignment of Computer Devices		
Atty Dckt No.: PSTM0002/MRK			

DECLARATION UNDER 37 C.F.R. 1.131

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Commissioner:

We, William W. Smith III, Charles D. Mentzer, Paul R. McLaughlin and Harland F. Maier Jr., do hereby declare and state as follows:

1. We, William W. Smith III, Charles D. Mentzer, Paul R. McLaughlin and Harland F Maier Jr., are co-inventors of the invention described and claimed in the above-referenced patent application. We each have reviewed the Office Action mailed October 21, 2003 in the above-referenced application. We have also reviewed the PR Newswire release by WorldWide Merchant regarding InterShipper 4.0 ("TEMPE"), and U.S. Patent Application Publication No. 2002/0022983 (Serial No. 09/873,756) ("*Barton*"). Both are relied upon by the Examiner as disclosing a computer system for managing shipping of a plurality of parcels by a plurality of users using a plurality of carriers recited in Claims 1-8, a method of configuring a plurality of server computer devices for managing shipping of a plurality of parcels by a plurality of users using a plurality of carriers r cited in Claims 9-19, and a computer program product embodying

computer program instructions for execution by a computer for configuring a plurality of server computer devices for managing shipping of a plurality of parcels by a plurality of users using a plurality of carriers recited in Claims 20-30. This Declaration is filed under 37 C.F.R. §1.131 to substantiate our invention of the subject matter claimed in the present application prior to March 23, 1999, the reported publication date of TEMPE, and therefore prior to April 30, 1999, the priority date indicated on the face of *Barton.*

2. Prior to March 23, 1999, we invented a multi-service, multi-carrier, Internet-enabled server-based shipping system for use by small volume shippers such as small businesses and home offices. The concept behind this multi-carrier, multi-service, Internet-based shipping system was to provide shipping users ("shippers") with a cross-comparison of shipment rating, service options, delivery schedules and other services by each of the multiple carriers for each of multiple services so that a shipper could compare multiple services offered by the multiple carriers and select one service offered by one of the multiple carriers to ship a parcel. When this shipping system was first conceived, we worked for MoveIt! Software Inc. ("MoveIt!"), a company founded in 1997. Later, MoveIt! became iShip.com, Inc., which eventually merged with Stamps.com Inc. and which is currently a wholly owned subsidiary of United Parcel Service ("UPS"). As of the latest date of this Declaration, iShip Inc. and Stamps.com are joint owners in common of the subject invention. Through all of the aforementioned corporate changes, we continued to develop and test the above-mentioned multi-carrier Internet-based shipping system that ultimately led to the filing of the above-referenced patent application.

3. As claimed in Claims 1 through 30 of the present application, as amended, the shipping management system, and the methods and computer program products for that system, that we invented prior to March 23, 1999, provided a shipping management system comprising a plurality of server computer devices. In support of the foregoing statement, we hereby submit, attached hereto as Exhibit A, a true and correct copy of pages 13-27 of a file copy of a Project Wolverine Design Document; we hereby submit, attached hereto as Exhibit B, a true and correct copy of pages 198-228 of a file copy the Project Wolverine Design Document. The Project Wolverine Design Document was kept confidential within MoveIt!. The Project Wolverine Design Document was prepared pursuant to an agreement with College Enterprises Inc. ("CEI") whereby MoveIt! was to install, operate, support, debug and nurture a Beta test version of an early prototype of the above-identified shipping system at a selected college campus. The purpose of the Beta test was to experiment with the early stage prototype system to determine if it worked, whether it would work over the Internet, and to identify and resolve problems and issues with the system. The CEI Beta Test Agreement provided for a

50/50 revenue sharing business model of gross profits. A true and correct copy of the CEI Beta Test Agreement with MoveIt! is attached hereto as Exhibit C. The finalization date of the Project Wolverine Design Document was prior to March 23, 1999.

4. As claimed in Claims 1 through 30 of the present application, as amended, the server-based, multi-carrier shipping management system, and the methods and computer program products for that system, that we invented prior to March 23, 1999 provided remote access by multiple shipping users via the Internet. In support of the foregoing statement, we hereby submit, attached hereto as Exhibit D, a true and correct copy of pages 33-34 of the Project Wolverine Design Document.

5. As claimed in one form or another in Claims 3, 7, 8, 13, 17, 24, 27 and 28 of the present application, as amended, the remotely accessible, server-based, multi-carrier, Internet-enabled shipping management system, and the methods and computer program products for that system, that we invented prior to March 23, 1999, provided for obtaining data from at least one system database in response to each user input of a request by each particular user to ship a parcel. In support of the foregoing statement, we hereby submit, attached hereto as Exhibit E, a true and correct file copy of pages 184-188 of the Project Wolverine Design Document, and attached hereto as Exhibit F, a true and correct copy of pages 288-298 of the Project Wolverine Design Document.

6. As claimed in Claims 4, 14, 18, 25 and 29 of the present application, as amended, the remotely accessible, server-based, multi-carrier, Internet-enabled shipping management system, and the methods and computer program products for that system, that we invented prior to March 23, 1999, provided for calculation of shipping rates for a plurality of carriers. In support of the foregoing statement, we hereby submit, attached hereto as Exhibit G, a true and correct file copy of Pages 140-146 of the Project Wolverine Design Document.

7. As claimed in one form or another in Claims 5, 6, 15, 19, 26, and 30 of the present application, as amended, the remotely accessible, server-based, multi-carrier, Internet-enabled shipping management system, and the methods and computer program products for that system, that we invented prior to March 23, 1999, provided for obtaining carrier tracking information from each of a plurality of carrier computer systems accessible over the global communications network. In support of the foregoing statement, we hereby submit, attached hereto as Exhibit H, a true and correct file copy of Pages 190-196 of the Project Wolverine Design Document, and attached hereto as Exhibit I, a true and correct copy of Pages 251-274 of the Project Wolverine Design Document.

8. From the time of our invention until the filing date on October 6, 1999 of a first provisional application on which priority for the present application is based in part, and

thereafter, we continued our effort to refine the operation of the system and resolve problems with the system. In support of the foregoing statement, we hereby submit, attached hereto as Exhibit J, a true and correct copy of a stream of emails exchanged between us, the inventors, and other members of the MoveIt!/IShip development team.

9. We hereby declare that all statements made herein of our own knowledge are true and that all statements made on information and belief are believed to be true; and further that the statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001, Title 18 of United States Code and that such willful false statements may jeopardize the validity of the application or any corresponding U.S. patent.

Date: April 2, 2004 _____
William W. Smith III

Date: April 21, 2004 _____
Charles D. Mentzer

Date: April 21, 2004 _____
Paul R. McLaughlin

Date: April 21, 2004 _____
Harland F. Maier, Jr.

EXHIBIT “A”

BEST AVAILABLE COPY

3. System Architecture

This section defines the system architecture for Project Gaucho. This definition includes a high-level explanation of the architectural specification for Gaucho and a detailed breakdown of the individual software and hardware components that comprise the fully functioning system.

3.1 Overview

Essentially, the architecture of the MoveIt! technology is organized into three (3) independent layers or architectural *tiers*. The first tier consists of *applications* that allow users to utilize MoveIt! features through their web browsers. The second tier contains *components* that perform general transaction management and analysis per the requests of the application tier. The third tier performs data management for the system including data storage and retrieval from the data base management system (DBMS), the file system, or a queue management facility. Figure 1 below illustrates the relationship of these tiers to the user and their web browser.

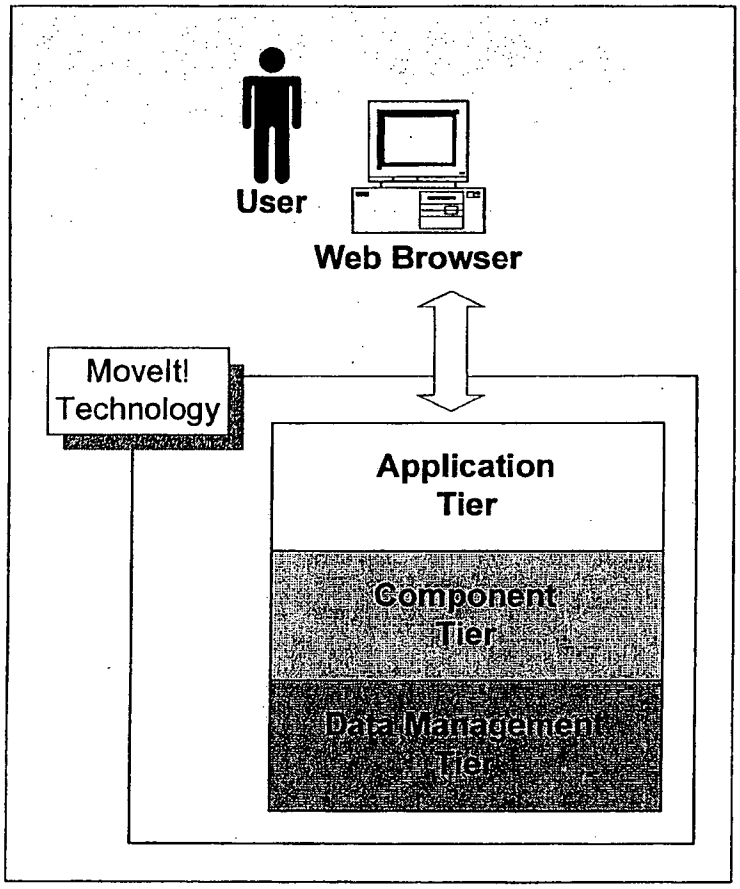


Figure 1. Gaucho Three-Tiered Architecture

MoveIt! supports both Microsoft and Netscape browsers. For the web client users, the Microsoft browser must be Internet Explorer 3.x (or more recent) and the Netscape browser must be Navigator 3.x (or more recent). For MoveIt! NOC client and Shipping Station client users the browser must be Microsoft Internet Explorer 4.0 (or more recent).

BEST AVAILABLE COPY

The three (3) tiered architecture can be further decomposed into separate components or partitions as illustrated in Figure 2 below. The *application* tier supports individual components that support the web client user, NOC operator, or shipping station user. Server components provide a variety of services to these clients. The data management tier is consists of components that manage data base storage, message queue storage and file storage.

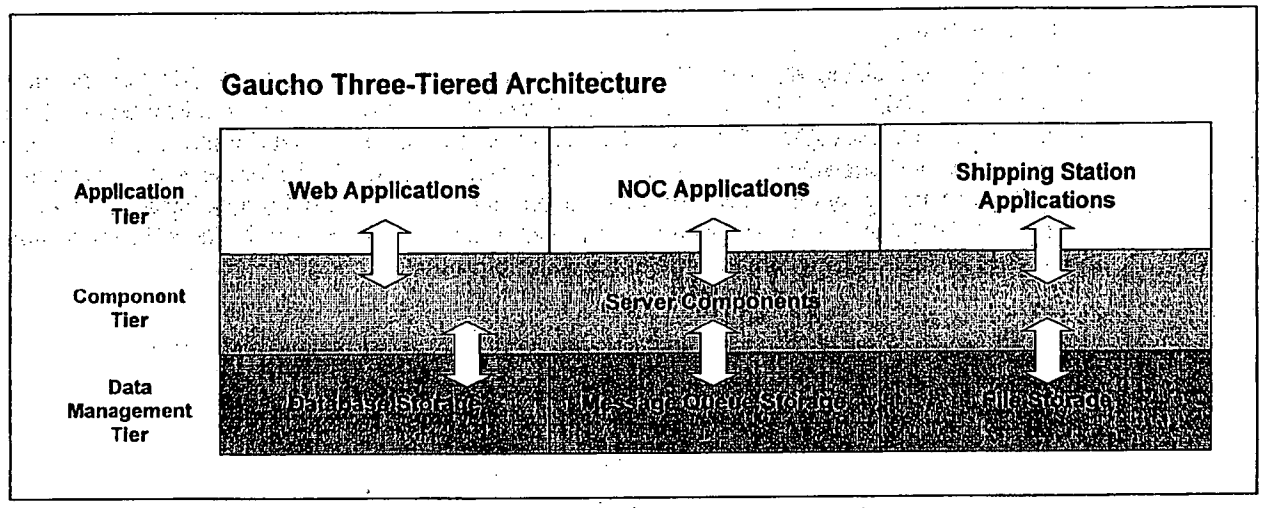


Figure 2. Gaucho Three-Tiered Architecture

Figure 3 below provides a detailed illustration of the application tier. Note that a client could utilize one or several applications in support of the user's browser activities.

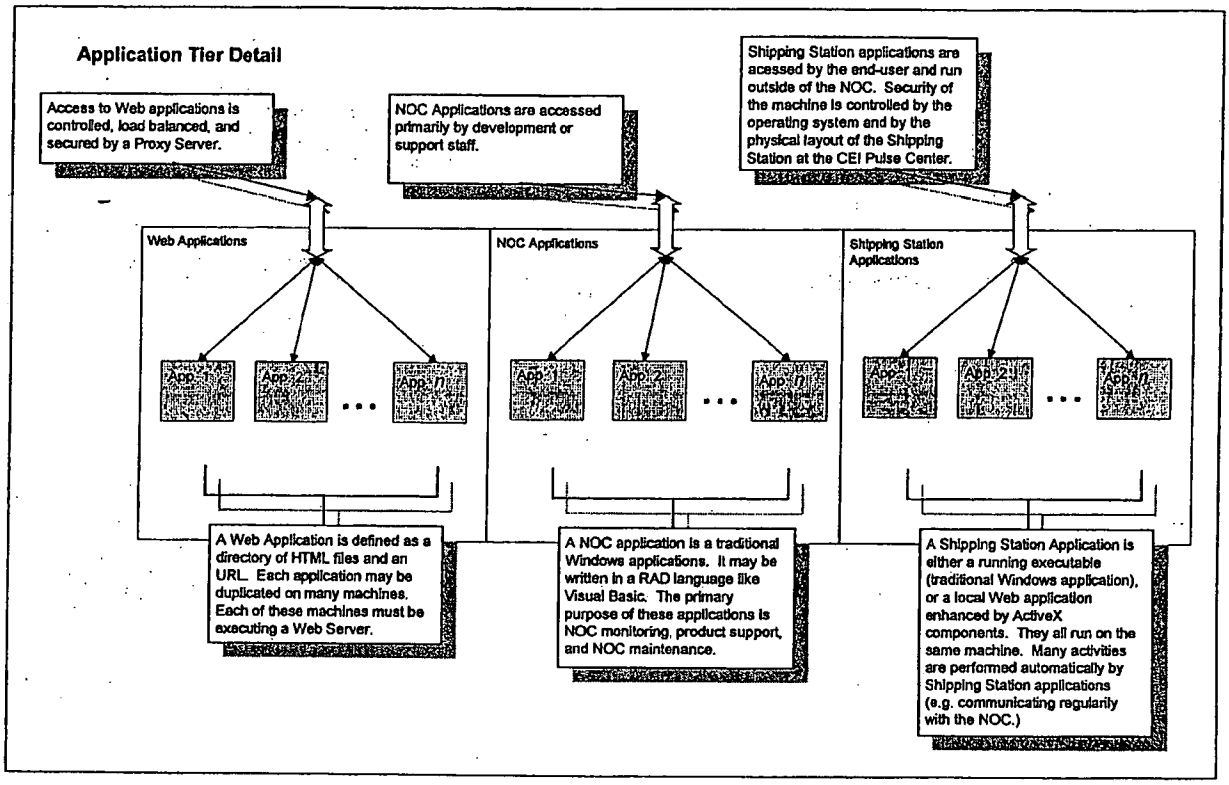


Figure 3. Application Tier Architecture Detail

Figure 4 below provides a detailed illustration of the component tier. Note that a given server component may operate within the context of the Microsoft Transaction Server (MTS) or it may be a separate (stand alone) executable component.

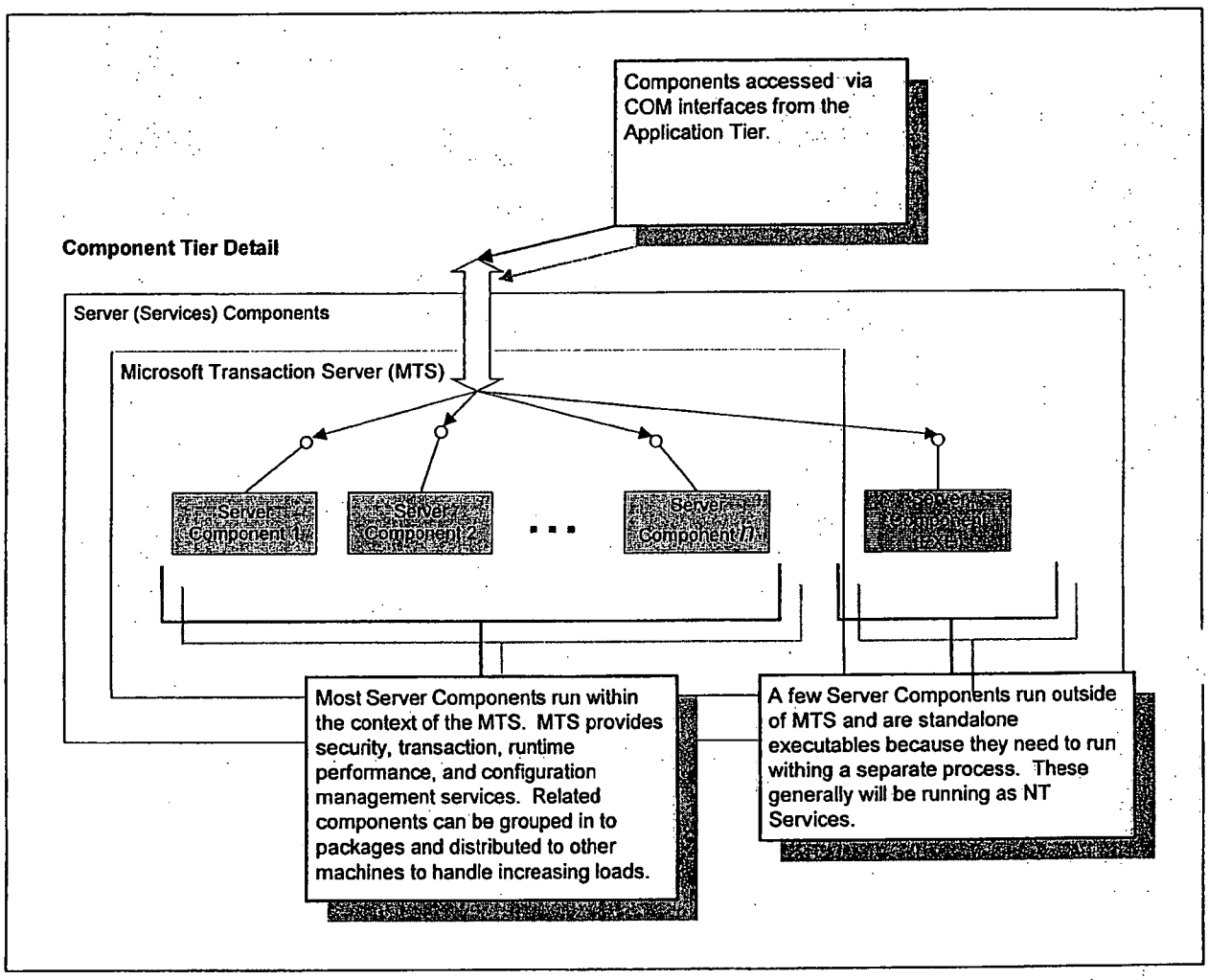


Figure 4. Component Tier Architecture Detail

Figure 5 below provides a detailed illustration of the data management tier.

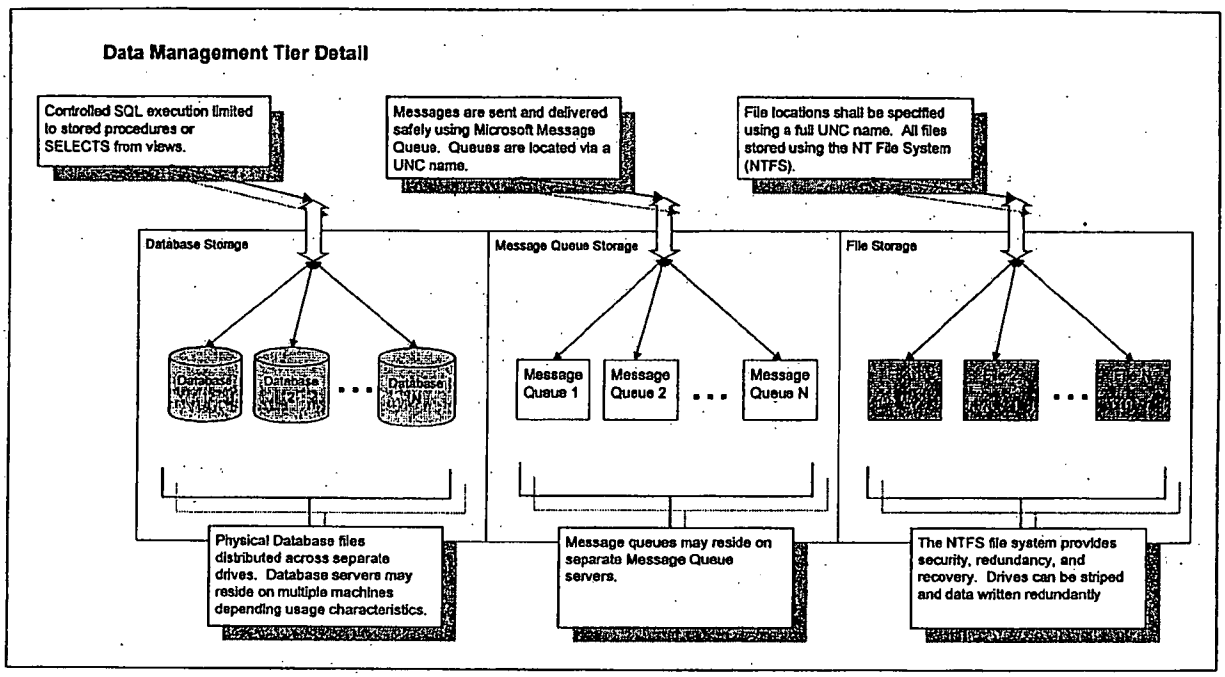


Figure 5. Data Management Tier Architecture Detail

3.2 Functional Areas And Tiers

To implement a particular functional requirement, end-user feature, or system service; requires the interaction of some or all of the tiers. The sections defines the mapping between Feature and Services to tiers. The details of the design presented in this document in alphabetical order. For each functional are described below in Table 3-1, there are Applications, and Server Components that implement these features. Following that, Table 3-2 defines the mapping between functional areas and architectual tiers.

Name	Description
Features	
Account Management	Supports end-user manipulation of their account preferences.
Address Book Management	Allows users to save commonly used shipping address for later use in shipping. At a later time users can modified their saved addresses.
Advertising	Web infrastructure, ASP code, etc that supports dynamic and static add placement in and around the major Web applications.
Billing	Provides functionality to allow reconcilliation of carrier bills with MoveIt! shipping manifest data.
Claims	Supports filing a claim against a manifested package.
History	Allows users to view selected portions of their shipping activity and perform operations on that data such as tracking.
Information	Web application that provides relatively static information about MoveIt! service offerings and small parcel shipping. Dynamic content should focus on FAQ or other technical support information, and survey questionnaires.
Registration	Application that collect information about a user, validates this data, and in general supports the creation of an account with the Wolverine system.
Reporting	Provides for generation of standards reports that reflect account activity and shipping history
Shipping	Web based small parcel shipping application.
Tracking	Web based small parcel tracking application. Implements interfaces that support multi-carrier tracking. Tracking in this version in an interactive request.
Services	
Business Rule	Validates and calculates shipping information. This component is the sole repository of small parcel shipping business rules.
Carrier Connectivity	Implements the communication link to the carrier back-end systems. Used by other services and applications such as Tracking.
Database Services	Data storage is provide by the data base either through ADO or ODBC. Database services provides a data dictionary object, and access to other shared resources related to the data base.
Drop Site	Manages drop site information and configuration data.
Event	Manages and dispatches messages and events. Reoccurring and one-shot events can be scheduled and dispatched from this service
Import/Export	Supports file import/export to and from the database. Multiple file formats are allowed.
Messaging	Provides store-and-forward messaging services. Used by other services such as carrier connectivity and by the Kiosk applications.
Rating	Calculates shipping rates based on simple package criterion.
Security	Responsible for validating user logins and for storing user privileges.
Shipping Station	Supports interfaces that the remote shipping station uses to interact with scales, label printers, and report printers.

Utility	Includes classes, and COM interfaces that are used by all other services. For example error logging classes, process control, configuration item storage and retrieval.

Table 3-1. System Features and Services.

Functional Areas	Applicable Tiers				
	Web Client Apps.	NOC Apps.	Shipping Station Apps.	Component Tier	Data Mangement Tier
Features					
Account Management				✓	✓
Address Book Management	✓			✓	✓
Advertising	✓			✓	✓
Billing		✓		✓	✓
Claims	✓			✓	✓
History	✓			✓	✓
Information	✓			✓	
Registration	✓				
Reporting	✓			✓	✓
Shipping	✓			✓	✓
Tracking	✓			✓	✓
Services					
Business Rule				✓	✓
Carrier Connectivity				✓	
Drop Site		✓		✓	✓
Event				✓	
Import/Export				✓	✓
Messaging				✓	
Rating				✓	✓
Security				✓	✓
Shipping Station			✓	✓	
Utility				✓	

Table 3-2. Functional Areas and Applicable Architecture Tiers.

3.3 Hardware Architecture

3.3.1 NOC Architecture

The Network Operations Center (NOC) is comprised of several types of servers configured into a high performance, reliable system. Table 6 below defines the six (6) types of servers contained within the NOC.

Server	Hardware	Software
NOC Web Applicaton Server (Configuration "W")	Dual Processor 200mhz Pentium Pro 256 MB Main Memory Dual 100MB Network Cards Dual Ultra-Wide SCSI Cards Four(4) 4GB Harddrives 5-10 Minute Full Power UPS	Microsoft NT Server 4.0 Microsoft Internet Information Server 4.0 (IIS 4.0) - Web And FTP Services Microsoft Systems Management Server 1.2 (SMS 1.2) - Network monitoring agent only.
NOC General Services Server (Configuration "G")	Dual Processor 200mhz Pentium Pro 256 MB Main Memory Dual 100MB Network Cards Dual Ultra-Wide SCSI Cards Four(4) 4GB Harddrives 5-10 Minute Full Power UPS	Microsoft NT Server 4.0 Microsoft Internet Information Server 4.0 (IIS 4.0) - Web And FTP Services Microsoft Systems Management Server 1.2 (SMS 1.2) - Network monitoring agent only. Microsoft Transaction Server 2.0 Microsoft Message Queue 1.0 - Primary or Backup Site Controller
Proxy Server (Configuration "P")	Single Processor 200mhz Pentium Pro 256 MB Main Memory Dual 100MB Network Card Ultra-Wide SCSI Cards Two(2) 4GB Harddrives 5-10 Minute Full Power UPS	Microsoft NT Server 4.0 - The proxy servers are the Primary & Secondary WINS servers. Also used as the Primary and Secondary Domain controllers. Network logins from the Shipping Stations would be controlled by the proxy machines. Microsoft Internet Information Server 4.0 (IIS 4.0) - Proxy Services Only Microsoft Proxy Server 2.0 Microsoft Systems Management Server 1.2 (SMS 1.2) - Network monitoring agent only.
NOC Monitoring And Adminstration Server (Configuration "N")	Single Processor 200mhz Pentium Pro 128 MB Main Memory Single 100MB Network Cards Ultra-Wide SCSI Cards Two(2) 4GB Harddrives 5-10 Minute Full Power UPS	Microsoft NT Server 4.0 Microsoft Systems Management Server 1.2 (SMS 1.2) - Monitoring Application SQL Server 6.5 - Storage for network monitoring data from SMS, and other application logs. Microsoft Mananagement Console 4.0
Database Management Server (Configuration "D")	Dual Processor 200mhz Pentium Pro 1GB Main Memory Dual 100MB Network Cards Dual Ultra-Wide SCSI Cards Four(4) 8GB Harddrives 5-10 Minute Full Power UPS Tape Backup	Microsoft NT Server 4.0 Microsoft Systems Management Server 1.2 (SMS 1.2) - Network monitoring agent only. Microsoft SQL Server 6.5 Microsoft Message Queue 1.0 - This machine is the Primary Enterprise Controller (PEC)

Figure 6. NOC Server Configurations

3.3.1.1 Alpha-1 (12/1/97 through 1/30/98)

Figure 7 below illustrates the NOC configuration for the alpha-1 period. This time period lasts approximately two (2) months.

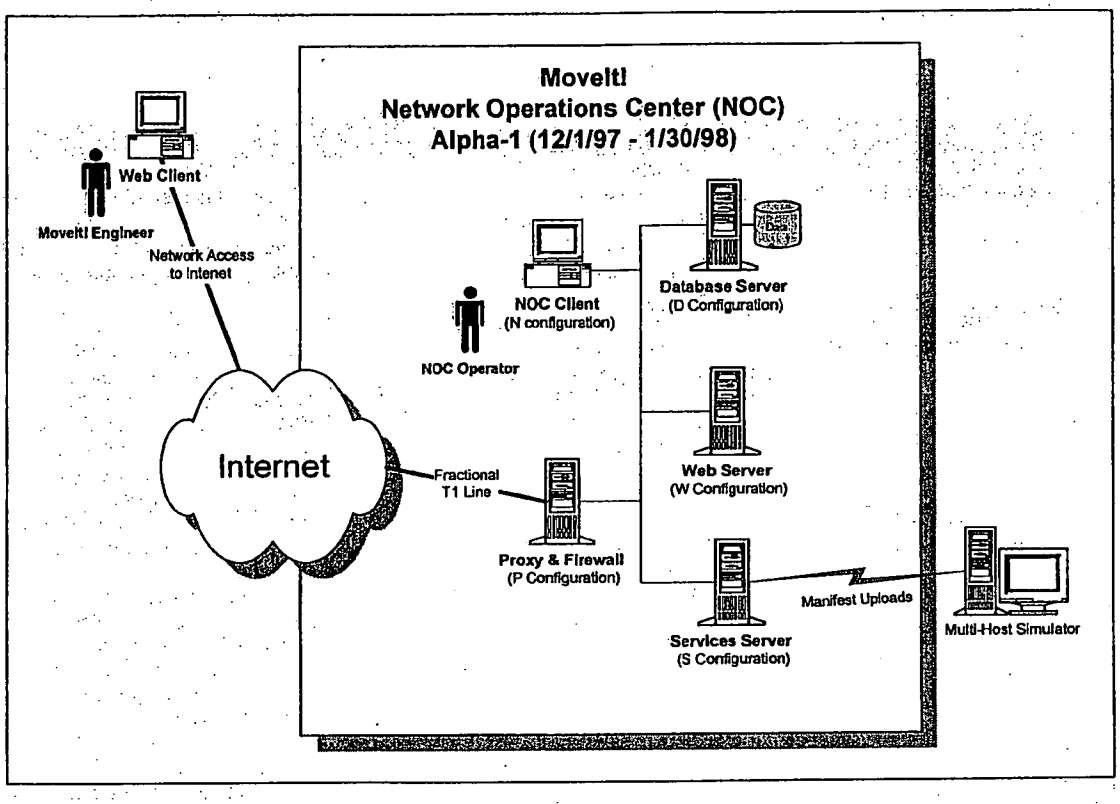


Figure 7. NOC Configuration (Alpha-1)

3.3.1.2 Alpha-2 (2/2/98 through 3/6/98)

Figur 8 below illustrates the NOC configuration for the alpha-2 period. This time period lasts approximately one month.

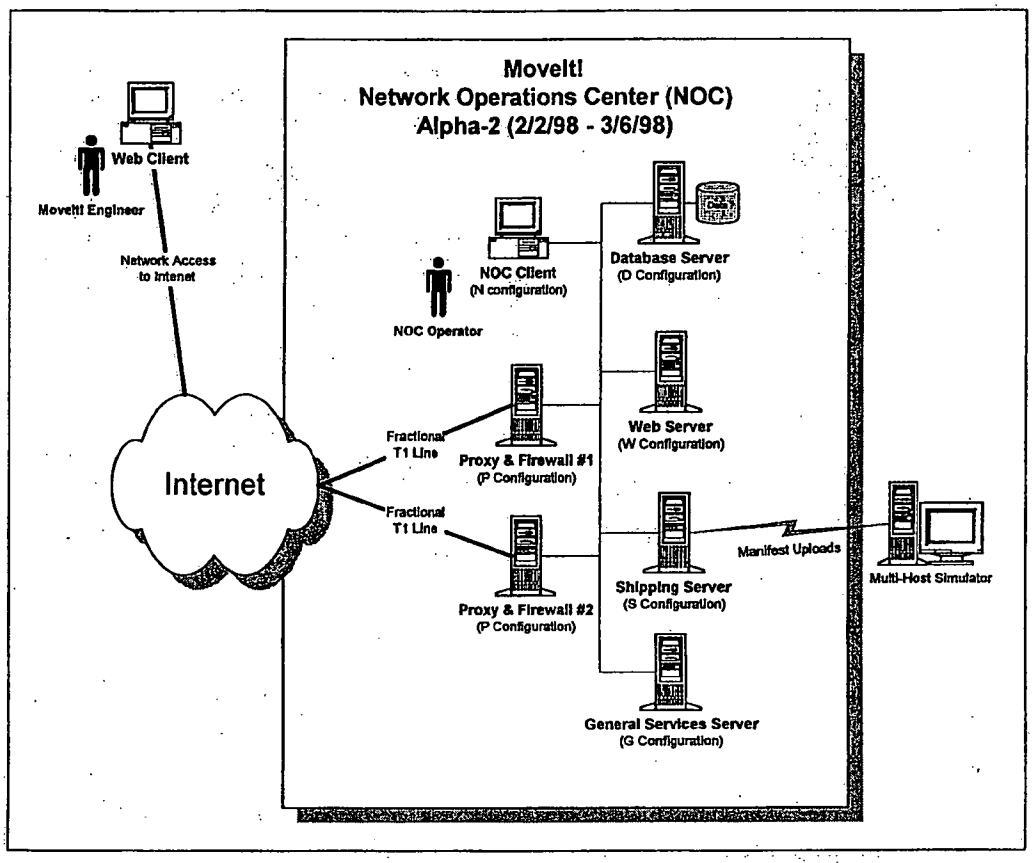


Figure 8. NOC Configuration (Alpha-2)

3.3.1.3 Alpha-3 (3/9/98 through 4/3/98)

Figure 9 below illustrates the NOC configuration for the alpha-3 period. This time period lasts approximately one month.

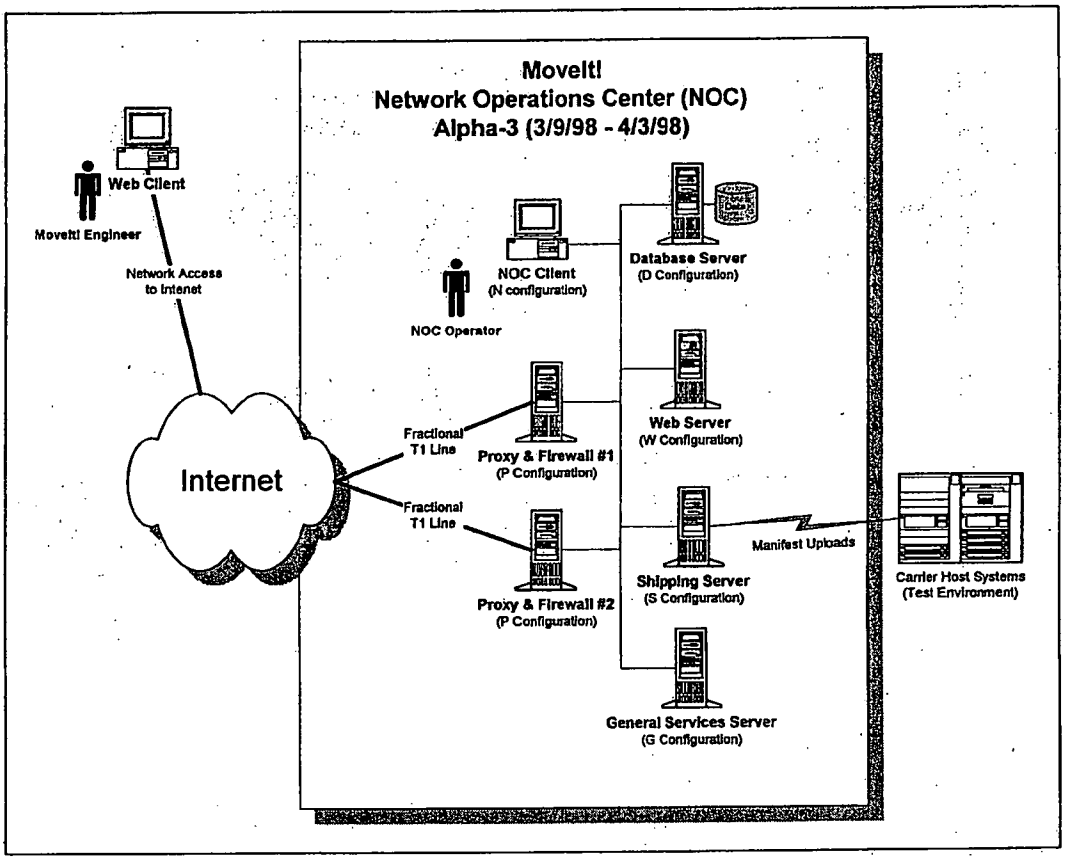


Figure 9. NOC Configuration (Alpha-3)

3.3.1.4 Beta (4/6/98 through 6/26/98)

Figure 10 below illustrates the NOC configuration for the Beta period. This time period lasts approximately three (3) months.

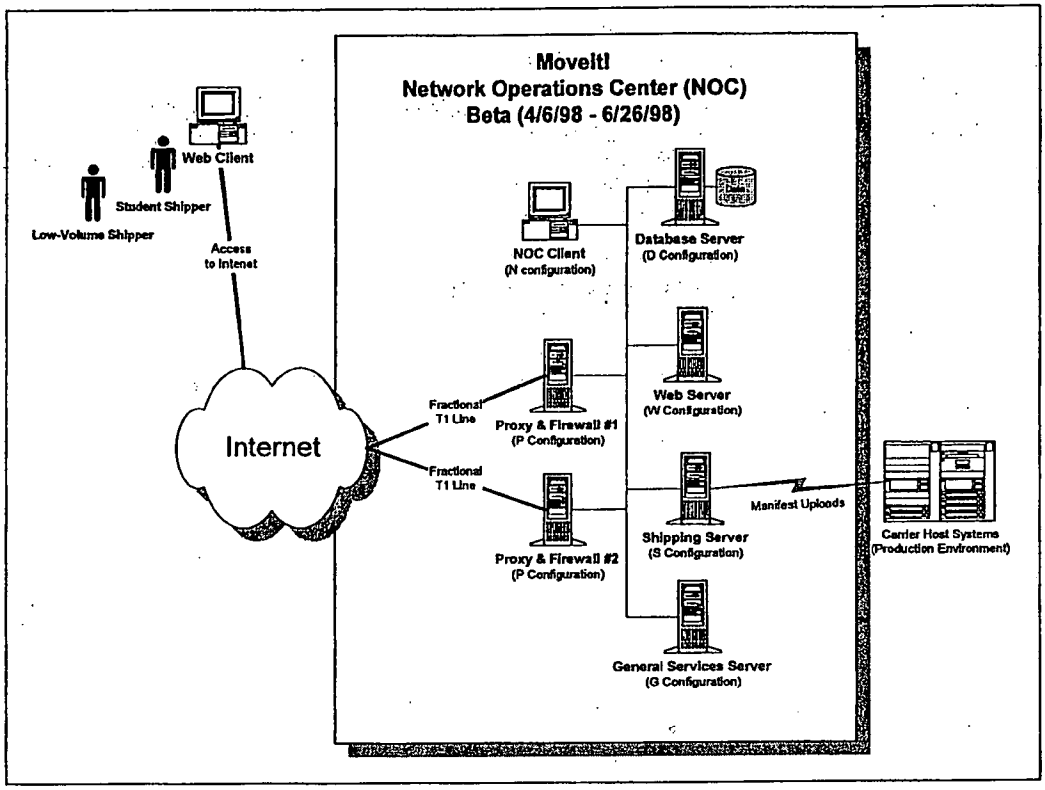


Figure 10. NOC Configuration (Beta)

3.3.1.5 Production 1.0 (6/27/98 through 11/29/98)

The production 1.0 system will be identical to the Beta system defined in the previous section.

3.3.1.6 Future Production Releases

TBD

3.3.2 Shipping Station Architecture

At each drop-off location there will be one(1) or more shipping stations. The shipping station is a Web applications agumented by ActiveX controls to support interaction with the connected hardware. Table 3 lists the hardware and software on the two(2) shipping station configurations. These shipping stations are connected a Web server at the MoveIt! NOC. Each shipping station must have an attached scale and a label printer. One of the shipping stations at the drop-off site must also have a report printer connected to it. This printer can be a simple dot matrix printer. The purpose of the report printer is to print summary manifests that the carriers require when picking up packages. The label printer must be able to printer both a receipt label a valid carrier shipping label. Figure 11 below illustrates the basic components of the shipping station.

Shipping Station	Hardware	Software
Standard Shipping Station (Configuration "S")	Single Processor 200mhz Pentium 32 MB Main Memory 56K Modem IDE or SCSI Controller One(1) 2GB Harddrive Power Filter ELTRON 2044 Digital Scale Capable of 150lbs.	Microsoft NT Workstation 4.0 Norton pcAnywhere Microsoft Internet Explorer 4.x
Full Shipping Station (Configuration "F")	Single Processor 200mhz Pentium 32 MB Main Memory 56K Modem IDE or SCSI Controller One(1) 2GB Harddrive Power Filter ELTRON 2044 Digital Scale Capable of 150lbs. EPSON Compatible Dot-Matrix Printer.	Microsoft NT Workstation 4.0 Norton pcAnywhere Microsoft Internet Explorer 4.x

Table 3. Shipping Station Configurations

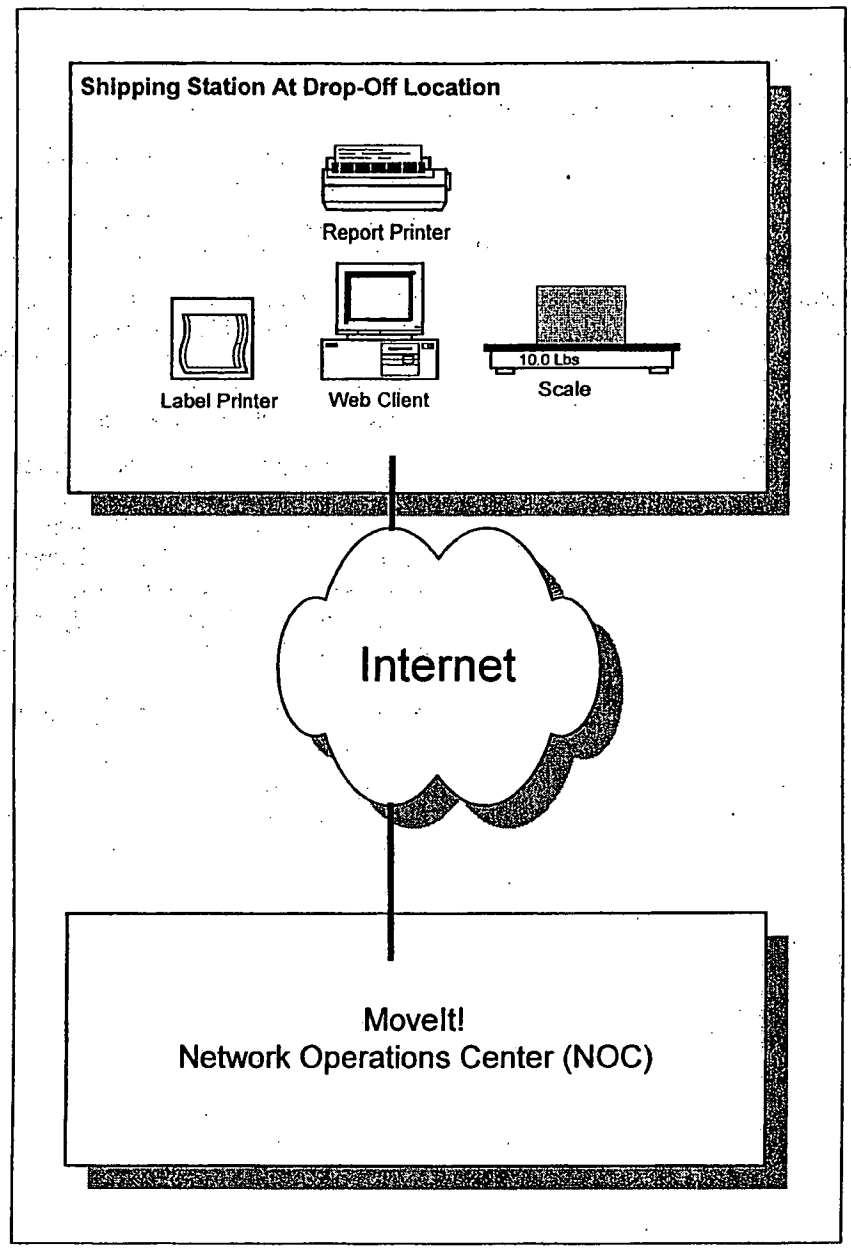


Figure 11. Shipping Station Architecture.

EXHIBIT “B”

9. Server Components Overview

A typical service component is a WIN32 DLL. It may export COM Classes, C++ Classes and/or Free Functions. Each component is different. Each detailed design section details what is exported to other components to use, the internal implementation details, and what requirements it fulfills.

9.1 Typical Component Design

Described below is a Design Pattern for a typical server component's COM interfaces. It takes into account security requirements, the use of the database for data storage, and usability of interfaces from scripting languages. All Server Components that use this pattern must support all the interfaces, properties, and methods described in this section unless they are noted as optional. (As you will see later in the detailed design sections for each component, most designs follow this pattern.) If a method does not apply to one of the standard interfaces a component is implementing, then the method coded to always return a failure code.

The pattern is a hierarchy of COM Interfaces. At the root of all the interfaces is the Service Interface. Essentially this interface is a secure class factory, through which useful interfaces are created. Security is based on a user's rights as defined in there current role. (Roles and security are defined in more detail in the Security Services design section.) These interfaces that can be created at this level are generally of two(2) types: Administrative Interfaces or Manager Interfaces. The Services interface may allow creation of more than one type of Administrative or Manager Interfaces.

The figure below shows the hierarchy of object in a typical Services Component. Each of these objects is accessed through a well defined interface. Italic characters indicate fields that are placeholders that would contain specific names for a specific component. For example, if the word *"Thing"* was replaced with "Package", PackageServices would be the name of highest level object. At the lowest level is a business object represented as *Thing* in this figure. It is through this object that most of the business rules of the system are enforced. The following sections describe the standard interfaces to each of these object.

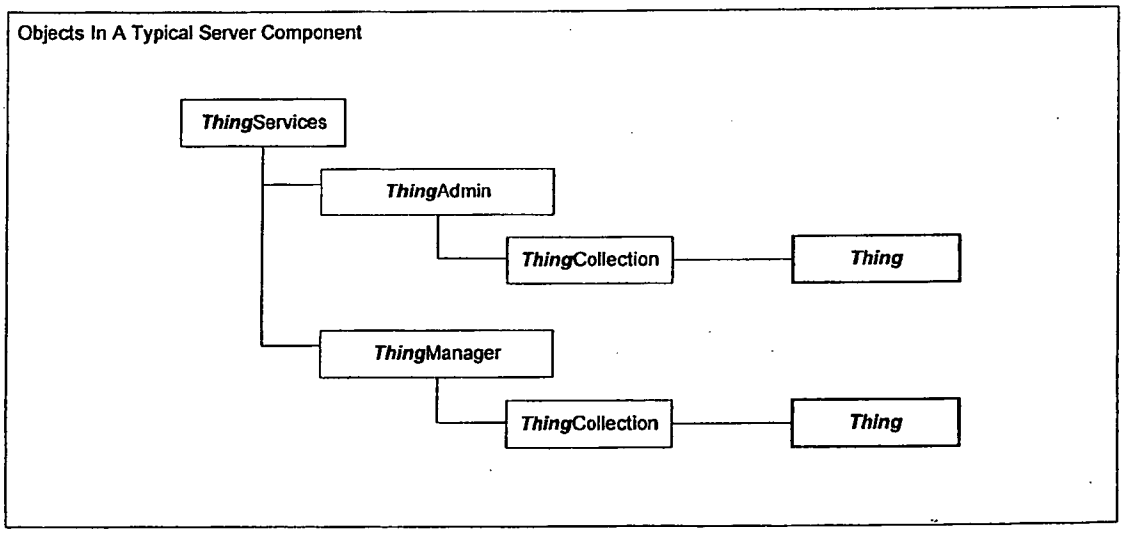


Figure 15. A Typical Server Component Object Model.

9.1.1 The Services Interfac

All service CoClasses must support IDispatch, IObjectControl, ISupportErrorInfo, and IUnkown. The IObjectControl interface is required specifically to support the Microsoft Transaction Server (MTS). One restriction imposed by MTS is that no component is allowed to support aggregation. Because some Services support more than one type of business object, more than one Manager and Admin Create() methods may be needed.

Attribute	Description
ProgID	MSI.*prefix_Thing*Services
Supported Interfaces	*prefix_IThing*Services IDispatch IObjectControl ISupportErrorInfo IUnknown

Table 5. COM Class *prefix_Thing*Services.

Interface: *prefix_IThing*Services	
Initialization/Un-initialization	
None are required.	
Methods	
Create*Thing*Manager()	Takes a secur_IUser inteface pointer. It returns a new *prefix_IThing*Manager interface. This interface is used by the general user. The user must be currently logged-on for the method to succeed and be in a role allowed for this service. A QueryInterface() will fail when trying to retrieve the *prefix_IThing*Manager interface.
Create*Thing*Admin()	Takes a secur_IUser inteface pointer and return a new *prefix_IThing*Admin interface. This interface is for administrators only. Contains methods that only NOC applications, NOC administrators or technical support can use. A QueryInterface() will fail when trying to retrieve the *prefix_IThing*Admin interface.
Properties	
None are required.	

Table 6. *Thing* Services Interface.

9.1.2 The Manag r Interfac

The purpose of the Manager interface is to control storage and retrieval of single or multiple business objects. The Manager interface makes no assumptions about where or how these business objects are stored. They may reside in an RDBMS, in a disk file, or in memory. Methods are subject to security checks against the current secur_IUser privileges.

Whether or not a filter string is an SQL clause or some other type of filter is dependent on the context and storage type. If the Manager supports SQL type clauses, lots of flexibility is possible, including changing the sort order. Read the documentation for the particular Manager you are accessing.

Interface *prefix*_I*Thing*Manager	
Initialization/Un-initialization	
None are required.	
Methods	
Items()	Returns a *prefix*_I*ThingCollection*. The number and ordering of *Things* in the collection is determined by the passed in filter and the current user.
Save()	Takes a *prefix*_I*Thing* as an argument and stores it in data storage. The Validate() method of *Thing* is called prior to any attempt to store the object. If the object is currently in data storage, it is updated otherwise it is inserted.
Load()	Based on an OID retrieves a single *Thing* object.
Find()	Based on a filter string, retrieves the first occurrence of a *Thing* object matching the filter. The returned *Thing* is NULL if nothing matches.
Count()	Based on a filter string, retrieves the number of a *Thing* objects matching the filter. The count is set to zero(0) if nothing matches.
Delete()	Deletes a *Thing* using the passed in OID.
Create()	Creates an empty thing. The properties are filled with all the property names and any default values are assigned. An OID is generated and the Name property is assigned at this time.
CreateFilterUsingExample()	Uses the properties of the passed in *prefix*_I*Thing* to build a filter string that is returned. Use the filter string on future calls to methods that require a filter.
Properties	
User	Returns the secur_IUser interface used during the creation of the prefix_*Thing*Manager.

Table 7. *Thing* Manager Interface.

9.1.3 The Collection Interface

The Items() method of the Manager interface returns a pointer to a Collection interface. This is a standard collection object as documented in the Microsoft Platform SDK. These objects allow retrieval of single items in the collection, or iteration of the entire collection using and enumerator. For more information of standard collection objects, see the Microsoft Platform SDK documentation.

Interface: *prefix*_I*Thing*Collection	
Identifier	
ProgID	MSI. *prefix*_I*Thing*Collection
Initialization/Un-initialization	
None are required.	
Methods	
Item()	Returns a *Thing* for the passed in index. Indexes can be numbers, strings, or other types. The Item() method may take one or more arguments to indicate the element within the collection to return. This method is the *default* member (DISPID_VALUE) for the collection object.
Properties	
_NewEnum	Returns a standard IEnumVARIANT interface. The number and ordering of *Things* that can be enumerated is determined by the current filter and the user's security level. You may issue another Items() call to the same *Thing* Manager and this collection remains valid. This method is marked "restricted" in the IDL. **Note:** Within the type library, the _NewEnum property has a special DISPID: DISPID_NEWENUM.
Count	The number of items in the collection. For a collection stored in the database, this method performs an SQL COUNT() statement. This is a "readonly" property.

Table 8. *Thing* Collection Interface.

9.1.4 The Enumeration Interface

The _NewEnum() method in the Collection interface returns the Enumeration interface documented in this section. This is a standard COM interface. (For more information on enumerators and the IEnumVARIANT interface, see the Microsoft Platform SDK documentation). Objects that support IEnumVARIANT can be used very simply by OLE Automation Controllers that support "FOR EACH ... NEXT" –like constructs. Behind the scenes the controller calls the _NewEnum property from the collection object and then does a QueryInterface to retrieve an IEnumVARIANT pointer. IEnumVARIANT::Next is then used to iterate over the collection and retrieve individual items in the collection. An example of an automation controller language is VBScript.

Interface IEnumVARIANT	
Initialization/Un-initialization	
None are required.	
Methods	
Clone()	Copies the current state of the enumeration so you can return to the current element after using Skip() or Reset()
Skip()	Skip over one or more elements in a collection
Reset()	Resets the current element to the first element in the collection
Next()	Retrieves one or more elements in a collection, starting with the current element
Properties	
<None are required>	
Operators	
<None are required>	

9.1.5 The Business Object Interface

Every business object shall implement the biz_IBusinessObject interface. If the object has specific methods then it should implement a separate interface with the name: *prefix_IThing*. The purpose of the business object interface is to allow consistent methods for data validation and property setting and retrieval. Having one consistent interface on all of the business objects in the system, allows very general purpose functions to be written that process business objects. An example is a presentation function that formats all of a business object's properties into HTML for display in a Web browser. This interface is defined in the Business Rule Services section of this design document.

An important property of this interface is that every business object has a unique id called on OID (Object Identifier). No two(2) new object will every have the same OID. This property of OIDs allows quick retrieval of objects from persistent storage, and also fast deletions.

9.2 Server Component Hierarchy

The figure below shows the hierarchy of physical dependencies between server component packages. At the bottom of the figure is the lowest level package (Utility). All other packages in the system are dependent on the facilities provided by this package. There are nine (9) levels in all. At the top are packages that implement the highest level features on the system such as Claims. Claims for example requires the facilities of Tracking and is therefore dependent upon it. The rule that must be followed is that a package may not use services provided by a package above it in the hierarchy.

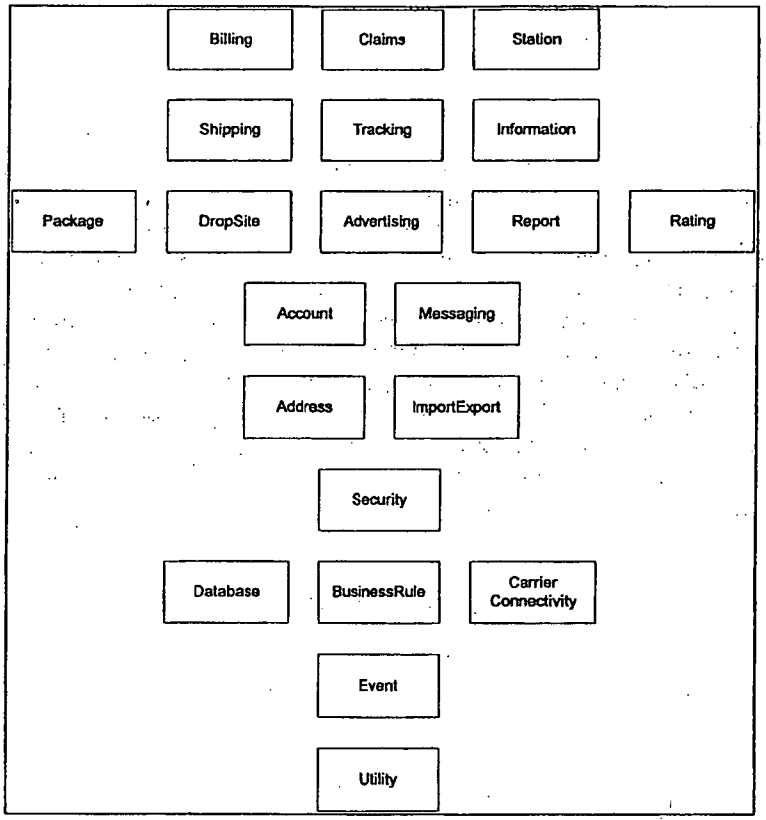


Figure 16. Server Component Hierarchy

9.3 Server Component Detail Designs

The following sections define in detail each of the server components.

9.3.1 Account Services

9.3.1.1 Overview

The Account Services provide methods for the administration of all information that is specific to an account or to a user. It is with the Account Services that accounts and users are created, altered, and deleted. These two entities which Account Services can alter are similar and sometimes contain duplicate information, but there are differences between the two. An account must contain at least one user while a user need not be connected to an account. The information contained within an account includes billing information, sub-accounts, and a list of account users. A user is an entity that has shipping information, assigned security roles, and other personal information. When a customer creates a new account, they are actually creating a new account and a new user, where the account has that newly created user. It is likely that the address contained within the account for billing is similar to the shipping address contained within the user entity. The account creator's user becomes the account administrator upon creation and can add new users to their account. The amount of information a user is allowed to administer depends on the security privileges they hold for the account.

An account administrator can perform the following abilities:

- Edit their user information
- Edit the account's information
- Create sub-accounts
- Create new users to be contained in the parent account or sub-accounts
- Administer security roles for the new users

Users who belong to the account may or may not have a number of these privileges. Most users start with the ability to alter their own information and then the account administrator can grants them more privileges.

The account services were designed to be flexible and easily maintained. The following specific items were kept in mind when designing the account services:

- Accounts can contain sub-accounts
- Users may belong to several accounts
- Users need the proper security role before they can administer account information
- Users have separate roles for separate accounts

Associated with users are security roles defined by Security Services. These roles are used to determine which Account Services interfaces a user may instantiate. Associated with accounts, users have separate security roles assigned to them which allow all other applications use to determine what actions a user may perform. A user might only hold a SECUR_GENERAL_USER security role when their secur_IUser interface is queried, but the user's acct_IUser interface may show a user as having a SECUR_ACCOUNT_ADMIN security role for a particular account.

9.3.1.2 Structure and Identification

The table below defines the naming prefix used to define objects within the Account Services and it defines the name of the component.

Attribute	Value
Prefix	acct
Name	Account
Type	DLL

Table 1. Account Structure and Identification.

9.3.1.3 Exported Interfaces

Exported interfaces are all classes, free functions, & COM interfaces that other packages may need to use. In other words, these are the services that this package is providing. Function arguments are described but not shown. This is done to make the document more readable and maintainable as the design evolves.

For the Account Services, there are no exported classes or free functions. There are five exported COM interfaces for performing various tasks. The main interface, which should be always accessible, is the acct_IAccountServices interface. With this interface, the other three Account Services' interfaces are retrieved. These interfaces include the accounts administration interface, the accounts interface, and the interface for accessing user information.

Classes

There are no exported classes for Account Services.

Free Functions

There are no free functions for Account Services.

COM Interfaces

All COM interfaces to the Account Services are channeled through the same interface. The acct_IAccountServices interface has methods that are used to create all other account interfaces. The ProgID that is stored in the registry and the interfaces that may be retrieved using it are listed in the table below.

Attribute	Description
ProgID	MSI.acct_AccountServices
Supported Interfaces	acct_IAccountServices IDispatch IObjectControl ISupportErrorInfo IUnknown

Table 2. COM Class Account Services

This section defines the COM interfaces that are exported from the Account Services. Each interface is defined in a table that describes its methods and properties.

Interface acct_IAccountServices

This interface is used to retrieve the other interfaces offered by the Account Services. It is instantiated without any need for initialization and will remain present for the life of the web server

service. Before any interface is created with acct_IAccountServices, a secur_IUser interface is needs to be passed. This security interface is queried to determine the security role a user has and depending on the security role found, different interfaces may be retrieved.

Interface: acct_IAccountServices	This interface provides a means to create the other account services interfaces.
Initialization/Un-initialization	
None are required.	
Methods	
CreateAccountAdmin()	This method receives a secur_IUser interface and returns a result and a interface for performing administration of all the accounts and users. This interface is used to manipulate Account Services' preferences.
CreateAccountManager()	This method is passed a secur_IUser interface and it returns a result and a interface for accessing the accounts to which the user belongs. Depending on security role the user holds, they can use methods to alter the account information.
Properties	
None are required.	

Table 3. acct_IAccountServices Interface Description.

Interface acct_IAccountAdmin

Only a user who is in the SECUR_ADMIN security role can retrieve this interface. With this interface, the administrative user can create, edit, or delete either accounts or users. It is only through this interface that accounts and users can truly be deleted. Other interfaces allow you to deactivate an account or remove a user from an account but they are not deleted from the database. Even the Registration Services must use this interface when it creates a new account for a new customer. Basically, all actions which can be performed on a user or account can be achieved through this interface.

Interface: acct_IAccountAdmin	This interface provides a means for creating and deleting accounts.
Initialization/Un-initialization	
None are required.	
Methods	
CreateAccount()	This method creates a new account object and returns a acct_IAccount interface. (note that the information has not been added to the database) If a parent account is specified, the new account is created as a sub account.
DeleteAccount()	This method is used to delete an account. It passed an account's object ID and returns a result.

ItemAccounts()	This method creates an acct_IAccountCollection interface. The number and ordering of Accounts in the collection is determined by the defined filter and the current user.
CountAccounts()	This method returns the number of all the possible enumerated accounts for the filter that was placed on the enumeration.
SaveAccount()	This method receives an acct_IAccount interface and it updates the database to reflect changes in the information. If no account ID is specified, a new account ID is created.
LoadAccount()	This method receives an account object ID and returns an acct_IAccount interface for the account.
FindAccount()	This method returns an acct_IAccount interface based on the defined filter and the user who is performing the find.
AddAccount()	This method receives two account object IDs and makes one account the parent of the other.
RemoveAccount()	This method sets the parent of a passed account object ID to be NULL.
CreateFilterUsingAccount()	This method receives an account interface and uses the defined properties as a filter for retrieving a collection of accounts.
CreateUser()	This method creates a new user object and returns the acct_IUser interface for it.
DeleteUser()	This method is used to delete a user. It is passed a user object ID for the user and returns a result.
ItemUsers()	This method creates an acct_IUserCollection interface. The number and ordering of Users in the collection is determined by the defined filter and the current user.
CountUsers()	This method returns the number of all the possible enumerated users for the filter that was placed on the enumeration.
SaveUser()	This method receives an acct_IUser interface and it updates the database to reflect changes in the information. Optionally, a list of possible user ID's are passed. The first unused user ID is used for this new user object.
LoadUser()	This method receives a user object ID and returns an acct_IUser interface.
FindUser()	This method retrieves an acct_IUser interface based on the defined filter and the user who is performing the find.
AddUser()	This method receives a list of user object IDs and an account object ID. The users becomes members of the account.
RemoveUser()	This method receives a list of user object IDs and an account object ID and it removes the users from the account. If this was the last account to which the user belonged, they will be removed from use.
CreateFilterUsingUser()	This method receives an user interface and uses the defined items as a filter for enumerating users.

Properties	
User()	Returns the secur_IUser interface used during the creation of the acct_IAccountAdmin.

Table 4. acct_IAdminAccounts Interface Description.

Interface acct_IAccountManager

With the acct_IAccountManager interface, the accounts to which a user belongs can be retrieved. If the user holds the SECUR_ACCOUNT_ADMIN security role for an account, they can use the methods that alter account information and user association. As an account administrator needs, they can add new users to an account and remove users who are no longer desired. When a user who only holds the SECUR_GENERAL_USER security role for an account uses this interface, they are not able to execute account-altering methods.

Interface: acct_IAccountManager	This interface allows the alteration of account information.
Initialization/Un-initialization	
None are required.	
Methods	
CreateAccount()	This method creates a new account object and returns a acct_IAccount interface. (note that the information has not been added to the database) A parent account must be specified and the new account is created as a sub account. Required account security role: SECUR_ACCOUNT_ADMIN
DeactivateAccount()	An account cannot be deleted from this interface but this method allows it to no longer be accessible by being passed the account object ID. Required account security role: SECUR_ACCOUNT_ADMIN
AddAccount()	This method receives two account object IDs and makes one account the parent of the other. Required account security role: SECUR_ACCOUNT_ADMIN
RemoveAccount()	This method sets the parent of a passed account object ID to be NULL. Required account security role: SECUR_ACCOUNT_ADMIN
ItemAccounts()	This method creates an acct_IAccountCollection interface. The number and ordering of Accounts in the collection is determined by the defined filter and the current user. Required account security role: SECUR_ACCOUNT_ADMIN
CountAccounts()	This method returns the number of all the possible enumerated accounts for the filter that was placed on the enumeration. Required account security role: SECUR_ACCOUNT_ADMIN

SaveAccount()	This method receives an acct_IAccount interface and it updates the database to reflect changes in the information. If no account ID is specified, a new account ID is created. Required account security role: SECUR_ACCOUNT_ADMIN
LoadAccount()	This method receives an account object ID and returns an acct_IAccount interface for the account.
FindAccount()	This method returns an acct_IAccount interface based on the defined filter and the user who is performing the find.
CreateFilterUsingAccount()	This method receives an account interface and uses the defined properties as a filter for retrieving a collection of accounts.
Properties	
User()	Returns the secur_IUser interface used during the creation of the acct_IAccountManager.

Table 5. acct_IAccountManager Interface Description.

Interface acct_IAccount

Any user who is a member of an account and holds a security role better than SECUR_RESTRICTED_USER for the account can retrieve this interface for their account. This interface allows for general account information to be retrieved and modified although modifications are stored in the database through this interface. Users who obtain this interface and hold a security role of SECUR_ACCOUNT_ADMIN for the account can save changes they've made to other users.

Interface acct_IAccount	**This interface allows the retrieval of account information**
Initialization/Un-initialization	
None are required.	
Methods	
GetProperties()	This method returns a map of the account information contained within this object.
PutProperties()	This method is passed a map of account information to be contained within this object.
GetProperty()	This method is passed a string and it returns a string of the corresponding account information that was contained within this object.
PutProperty()	This method is passed two strings, one of which is a label and one is the matching value to be contained within this object.
IsPropRequired()	This method passed a property string and returns a Boolean value that tells if the property is required.
IsPropEdittable()	This method passed a property string and returns a Boolean value that tells if the property can be altered.
IsPropDirty()	This method passed a property string and returns a Boolean value

	that tells if the property has been altered with information other than that found in the database.
IsDirty()	This method returns a Boolean value that tells if the object has been altered with information other than that found in the database.
IsFromStorage()	This method returns a Boolean value that tells if the object was originally retrieved from the database or it was created new.
IsValid()	This method returns a Boolean value which informs whether all information contained within the object is valid. A map of invalid items are returned if any exist.
Validate()	This method forces all the invalid items to validate if possible.
GetOID()	This method returns the unique object identifier for the account object.
CreateUser()	This method creates a new user object and returns the acct_IUser interface for it. Required account security role: SECUR_ACCOUNT_ADMIN
DeactivateUser()	A user cannot be deleted from this interface but this method allows it to no longer be accessible by being passed the user object ID. Required account security role: SECUR_ACCOUNT_ADMIN
ItemUsers()	This method creates an acct_IUserCollection interface. The number and ordering of Users in the collection is determined by the defined filter and the current user. Required account security role: SECUR_ACCOUNT_ADMIN
CountUsers()	This method returns the number of all the possible enumerated users. Required account security role: SECUR_ACCOUNT_ADMIN
SaveUser()	This method receives an acct_IUser interface and it updates the database to reflect changes in the information. Optionally, a list of possible user ID's are passed. The first unused user ID is used for this new user object. Required account security role: SECUR_ACCOUNT_ADMIN
AddUser()	This method receives a list of user object IDs and an account object ID. The users becomes members of the account. Required account security role: SECUR_ACCOUNT_ADMIN
RemoveUser()	This method receives a list of user object IDs and an account object ID and it removes the users from the account. If this was the last account to which the user belonged, they will be removed from use. Required account security role: SECUR_ACCOUNT_ADMIN
LoadUser()	This method receives a user object ID and returns an acct_IUser interface.
FindUser()	This method retrieves an acct_IUser interface based on the defined filter and the user who is performing the find.

CreateFilterUsingUser()	This method receives an user interface and uses the defined items as a filter for enumerating users.
Properties	
User()	Returns the secur_IUser interface used during the creation of the acct_IAccount.

Table 6. acct_IAccount Interface Description.

Interface acct_IUser

This interface allows a user to retrieve or update their user specific information. When changes are made to a user object, they are not reflected in the database until another interface performs a save on the object. When a new user object is created, no user ID is specified for it. Before the user object is saved for the first time, a list of possible user IDs is passed and the first unused user ID is given to the object.

Interface acct_IUser	This interface allows for a users information to be reviewed or updated.
Initialization/Un-initialization	
None are required.	
Methods	
GetProperties()	This method returns a map of the user information contained within this object.
PutProperties()	This method is passed a map of user information to be contained within this object.
GetProperty()	This method is passed a string and it returns a string of the corresponding user information that was contained within this object.
PutProperty()	This method is passed two strings, one of which is a label and one is the matching value to be contained within this object.
IsPropRequired()	This method is passed a property string and returns a Boolean value that tells if the property is required.
IsPropEdittable()	This method passed a property string and returns a Boolean value that tells if the property can be altered.
IsPropDirty()	This method passed a property string and returns a Boolean value that tells if the property has been altered with information other than that found in the database.
IsDirty()	This method returns a Boolean value that tells if the object has been altered with information other than that found in the database.
IsFromStorage()	This method returns a Boolean value that tells if the object was originally retrieved from the database or it was created new.
IsValid()	This method returns a Boolean value which informs whether all information contained within the object is valid. A map of invalid

	items are returned if any exist.
Validate()	This method forces all the invalid items to validate if possible.
GetOID()	This method returns the unique object identifier for the user object.
IsUserInRole()	This method returns the security role which the user currently has associated with the account which instantiated this interface.
Properties	
User()	Returns the secur_IUser interface used during the creation of the acct_IUser.

Table 7. acct_IUser Interface Description.

Interaction Diagrams

To better understand the interactions with Account Services and other package and components of project Wolverine, interaction diagrams are presented. The diagrams do not depict what is going on inside of the Account Services for it is treated here as a black box.

Creation of New Account Interaction Diagram

The following figure shows the interactions between a Web Server Application and the Account Services' interfaces for creating a new account. The account created is a sub-account to another of the user's accounts. The diagram depicts a simple error free scenario. In the first step, the acct_IAccountServices interface is used to create an interface to an acct_IAccounts interface. This interface is used to enumerate the accounts that the user can administer. The next scenario begins when the user chooses an account under which to create the sub-account. The new account is created and filled with default properties which are used to fill the form which the user is offered. Finally, after the user fills in the form, the properties are placed within the account, the account is validated, and the account is stored into the database.

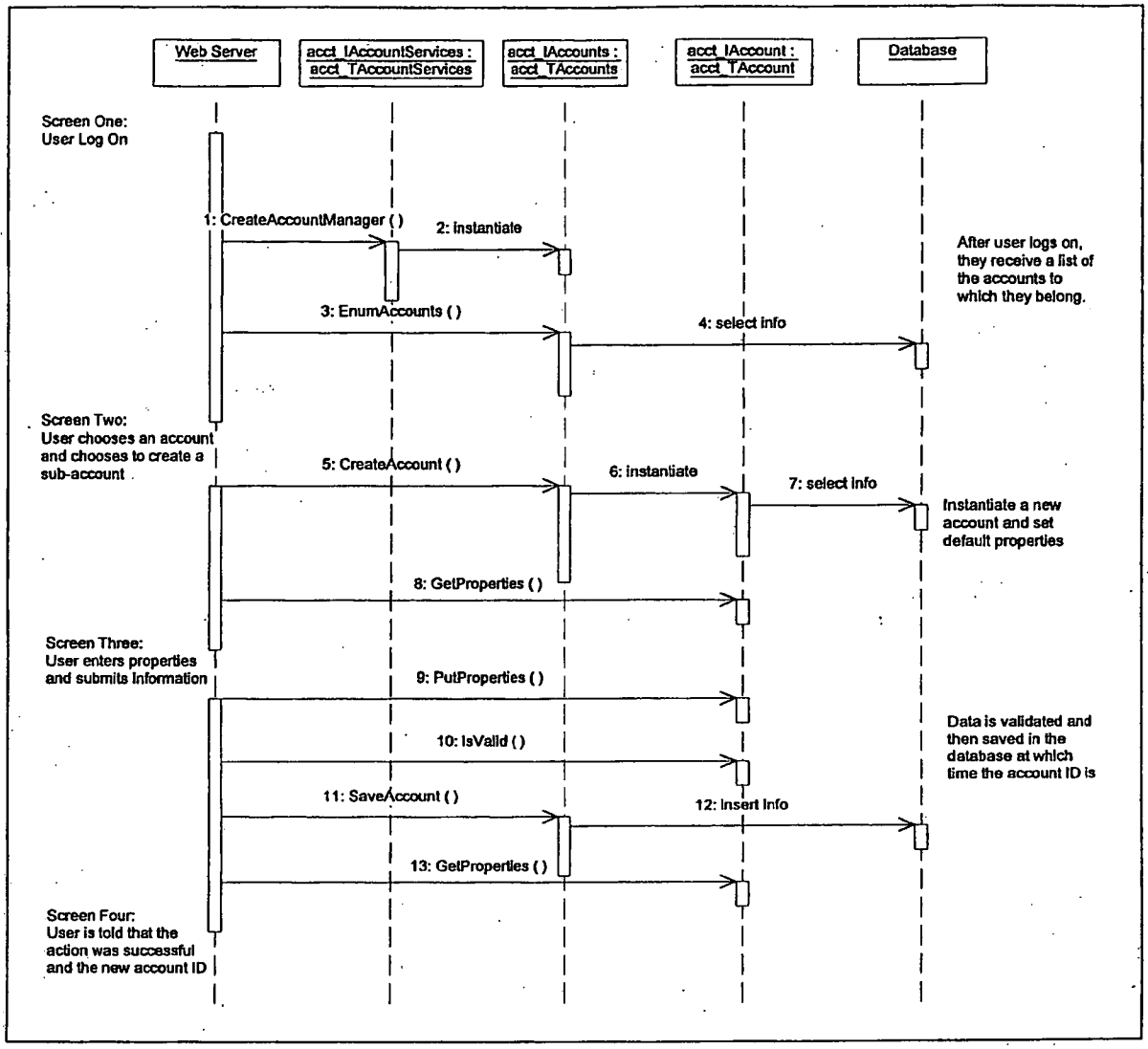


Figure 1. Creation of New Account Interaction Diagram

Addition of New User Interaction Diagram

The following figure shows the interactions between a Web Server Application and the Account Services' interfaces for creating a new user in an account. The user created becomes a member of one of the accounts that the account administrator can administrate. The diagram depicts a simple error free scenario. In the first step, the acct_IAccountServices interface is used to create an interface to an acct_IAccounts interface. This interface is used to enumerate the accounts that the user can administer. The next scenario begins when the user chooses an account under which to create the new user. A new user is created and filled with default properties which are used to fill the form which the administrator is offered. Finally, after the administrator fills in the form, the properties are placed within the new user, the user is validated, and the user is stored into the database.

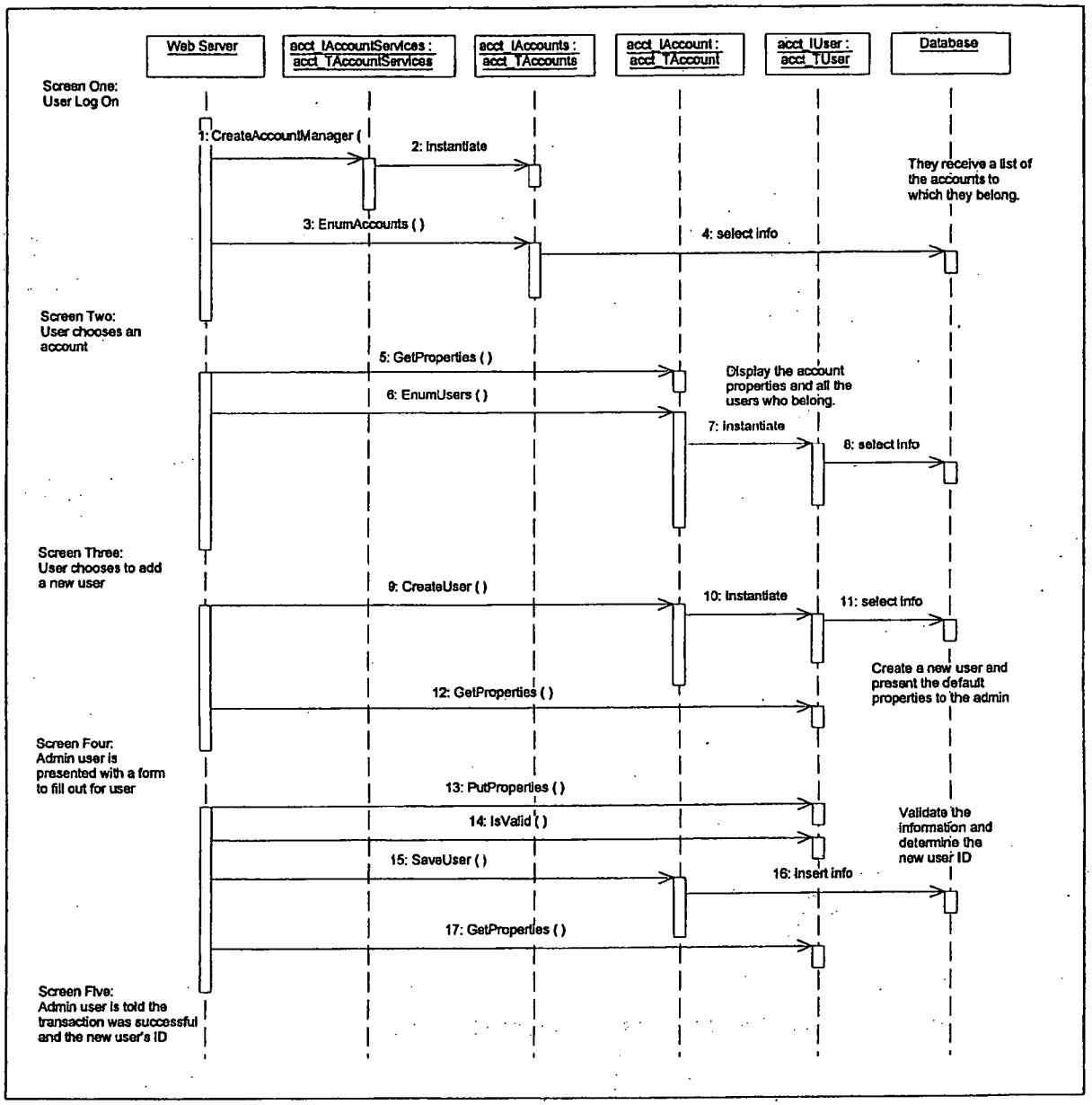


Figure 9.3.1-2. Addition of New User Interaction Diagram

Messages

The table below defines messages that either <Gadget> Services publishes or subscribes to. For more information on messages, see the Event Services design section in this document.

Name	Description
STATUS_INFO	
< PKG_IGADGET.PERFORMING_INITIALIZATION >	<Message sent at the start of initialization.>

_STATUS:WARNING	
<None are required.>	
_STATUS:ERROR	
< PKG_IGADGET.DB_CONNECTION_FAILURE >	<Couldn't connect to the database during the initialization process>
_JOB	
< PKG_IGADGET.DO_SOMETHING>	<Performs DoSomething() in response to the job request message.>

Table 8. Publish and Subscribe Messages Supported by <Package Name> Services.

9.3.1.4 Implementation Details

Under each of the external interfaces is a supporting class. The functionality of the methods provided by each of the classes is described by the corresponding COM interfaces listed above.

Classes

Class Diagrams

The classes used to implement the Account Services have few interactions. They are tied together because some of the classes are used to create other class objects. For instance, the acct_TAccountServices class has a method for creating an acct_TAdminAccounts object. Although the class object is created by acct_TAccountServices, the responsibility for freeing the object lies elsewhere. A number of utility classes are used by Account Services to perform various canned functionality. A few are shown as examples in the below diagram.

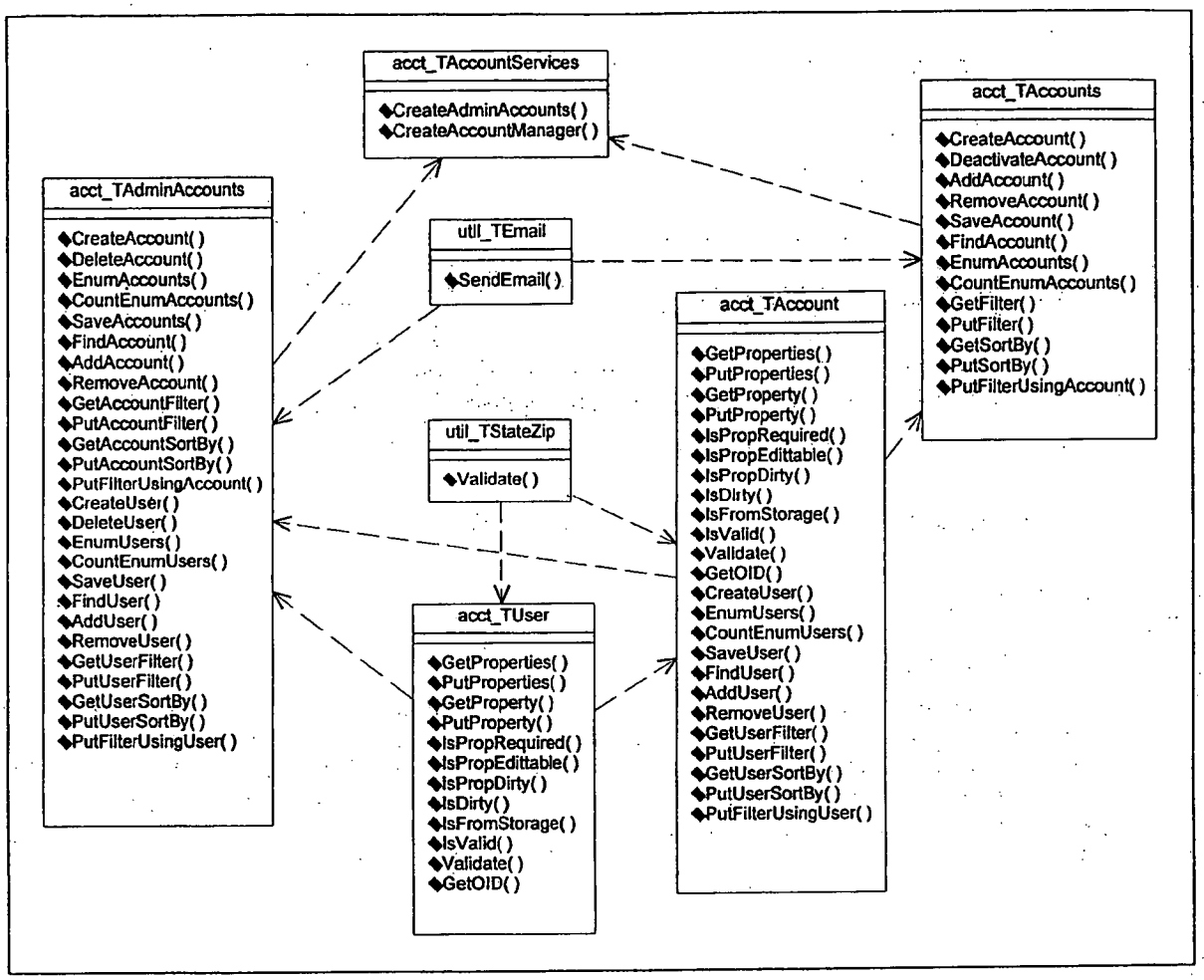


Figure 3. Class Hierarchy Diagram for Account Services.

Free Functions

There are no free functions required for the Account Services.

Interaction Diagrams

There are no interaction diagrams required for the Account Services.

9.3.1.5 External Dependencies

The Account Services relies on a number of database tables to contain the information about accounts and users. The associations between users and accounts are contained in the database but the rules applied to their association are mainly stored within the Account Services' classes.

The table below lists dependencies that this package has on other components and database tables.

Type	Name	Description
Database Table	vw_Account	Account Information
	vw_User	User Information

	vw_AccountAndUser	Users belong to Accounts
Utility Package Classes	util_TEmail util_TStateZip	Sending email Validation of state and zip code
COM Object	IDictionary IList	Microsoft dictionary Microsoft list
DLL	MFC42.DLL	Microsoft MFC support DLL
Static Library	ATL	Active X Template Library >

Table 9. Account Services External Dependencies

9.3.1.6 Requirements Traceability Matrix

< The table should list all the individual line item requirements met by the application. The Class/Function/Interface column should list the names of these objects.>

The table below traces requirements from the Server functional requirements to components of the < Package Name > service.

Requirement ID	Class/Function/Interface	Comments
<1.1.1>	<pkg_IGadget>	<Really, it does fulfill the requirement!>

Table 10. <Package Name> Services Requirements Traceability Matrix

9.3.2 Address Book Services

9.3.2.1 Overview

Address Book Services provides users with the capabilities to maintain their own recipient address in the database.

The Address Book Services package is designed to allow the user to:

- Add a new recipient address to the database.
- Update an existing recipient address in the database.
- Delete an existing recipient address from the database.
- Retrieve recipient addresses from the database.

9.3.2.2 Structure and Identification

The table below defines the naming prefix used to define objects within the Address Book Services component and it defines the name of the component.

Attribute	Value
Prefix	addr
Name	Address
Type	DLL

Table 11. Address Book Structure and Identification.

9.3.2.3 Exported Interfaces

Exported interfaces are all classes, free functions, & COM interfaces that other packages may need to use. In other words, these are the services that this package is providing. Function arguments are described but not shown. This is done to make the document more readable and maintainable as the design evolves.

9.3.2.3.1 Classes

None are required.

9.3.2.3.2 Free Functions

None are required.

9.3.2.3.3 COM Interfaces

This section defines the COM interfaces that are exported from the Address Book Services. Each interface is defined in a table that describes its methods and properties.

COM Class AddressBookServices

Attribute	Description
ProgID	MSI.addr_AddressBookServices
Supported Interfaces	addr_IAddressBookServices IDispatch IObjectControl ISupportErrorInfo IUnknown

Interface addr_IAddressBookServices

This main interface will create the address book interface manager addr_IAddressManager.

Interface: addr_IAddressBookServices	
Identifier	
ProgID	MSI.addr_IAddressBookServices
Initialization/Un-initialization	
Activate()	Performs context-specific initialization.
Deactivate()	Releases the object's ObjectContext reference and does other context-specific cleanup.
Methods	
CreateAddressManager()	Takes a secur_IUser inteface pointer and returns a new addr_IAddressManager interface. This interface is used by the general user. The user must be currently logged-on for the method to succeed and have the correct privilege.
CreateAddressAdmin()	Takes a secur_IUser inteface pointer and return a new addr_IAddressAdmin interface. This interface is for administrators only. Contains methods that only NOC applications, NOC administrators or technical support can use. A QueryInterface() will fail when trying to retrieve the addr_IAddressAdmin interface.
Properties	
None are required.	

Table 12. addr_IAddressBookServices Interface Description.

Interface addr_IAddressManager

The Manager interface looks like a standard collection interface to scripting languages. Methods are subject to security checks against the current secur_IUser privileges.

Interface addr_IAddressManager	
Identifier	
ProgID	MSI.addr_IAddressManager
Initialization/Un-initialization	
None are required.	
Methods	
Items()	Retuns an addr_IAddressCollection. The number and ordering of addr_IAddress in the collection is determined by the current filter, sort by properties, and the user's security level.
Save()	Takes an addr_IAddress as an argument and stores the address book data in database. The Validate() method of addr_IAddress is called prior to any attempt to store the object. If the object is currently in database, it is updated otherwise it is inserted.
Find()	Based on an OID creates and retrieves a single addr_IAddress.
Delete()	Deletes an addr_Iaddress using the passed in OID.
Create()	Creates an empty addr_IAddress. The properties are filled with all the property names and any default values are assigned. An OID is also generated and assigned at this time.
PutFilterUsingExample()	Uses the properties of the passed in addr_IAddress to build an SQL where clause filter. The Filter property is then set. Effects the results of the next call to retrieve a collection.
Properties	
GetFilter()	Returns the current filter string.
PutFilter()	Sets the filter string. The string is an SQL where clause minus the "WHERE". Effects the results of the next call to retrieve a collection.
GetSortBy()	Returns the current sort by string.
PutSortBy()	Set the order by string. The string is an SQL order by clause minus the "ORDER BY". A default order by shall be set when addr_IAddress interface is created. Effects the results of the next call to retrieve a collection.

GetUser()	Returns the secur_IUser interface used during the creation of the addr_IAddressManager.

Table 13. addr_IAddressManager Interface Description.

Interface addr_IAddr ssCollection

This interface is a collection of address books. It supports three read-only properties, each of which has a single accessor function.

Interface: addr_IAddressCollection	
Identifier	
ProgID	MSI.addr_IAddressCollection
Initialization/Un-initialization	
None are required.	
Methods	
Item()	Returns an addr_IAddress for the passed in index.
Properties	
_NewEnum()	Returns an IEnumVARIANT interface. The number and ordering of addr_IAddress that can be enumerated is determined by the current filter, sort by properties, and the user's security level. If you are enumerating things and the filter is changed, and error will occur when the next thing is retrieved. This method is marked "restricted" in the IDL.
Count()	The number of items in the collection.

Table 14. addr_IAddressCollection Interface Description.

Interface addr_IAddress

This interface allows Web server to interrogate the address book properties.

Interface: addr_IAddress	
Identifier	
ProgID	MSI.addr_IAddress
Initialization/Un-initialization	
None are required.	
Methods	
Validate()	Returns a boolean value indicating whether some property was found to be invalid. If one or more properties are invalid an additional argument populates a Map with property names and error text. Only returns a bad HRESULT if a fatal error has occurred. This method is responsible for inter-field validation. It is assumed that the properties are valid
GetProperties()	Returns an IMap interface of properties.
PutProperties()	Takes an IMap interface. Properties are validated for data dictionary violations but not for inter-property validation. The OID cannot be changed using this method.
GetProperty()	Returns the value of a single property as a VARIANT. The property is indexed by a key.
PutProperty()	Set the value of a single property. Requires a key and VARIANT property data. The property is validated for data dictionary violations. The OID cannot be changed using this method.
GetPropValues()	Takes a property name and returns a VARIANT array of allowed values. Used for example to populate a list box.
IsPropRequired()	Takes a property name and returns a boolean value indicating whether the property is required to have a value.
IsPropEditable()	Takes a property name and returns a boolean value indicating whether the property is available for editting, otherwise, the property is read-only.
IsPropDirty()	Takes a property name and returns a boolean value indicating whether the property has been modified since the last successful validation.
IsDirty()	Returns a boolean value indicating whether any property has been modified since the object was created.
IsValid()	Returns a boolean value indicating whether a successful validation had been performed since the last property

	change. Does not call the Validate() method.
IsFromStorage()	Returns a boolean value indicating whether the object was originally retrieved from data storage. This is used when saving an object back into storage to determine if an Insert or Update needs to be performed.
Properties	
GetOID()	Retrieves the OID of the current object.

Table 15. addr_IAddress Interface Description

9.3.2.3.4 Interaction Diagrams

To better understand the interactions with Address Book Services and other package and components of project Wolverine, interaction diagrams are presented. Each diagram depicts a scenario showing one or more components interacting. The diagrams do not depict what is going on inside of the Address Book Services. It is treated here as a black box.

Figure 4. shows how the interaction between a Web Server Application and the Address Book Services interfaces. The diagram depicts a simple error free scenario. It demonstrates how a new recipient address is created and insert into database.

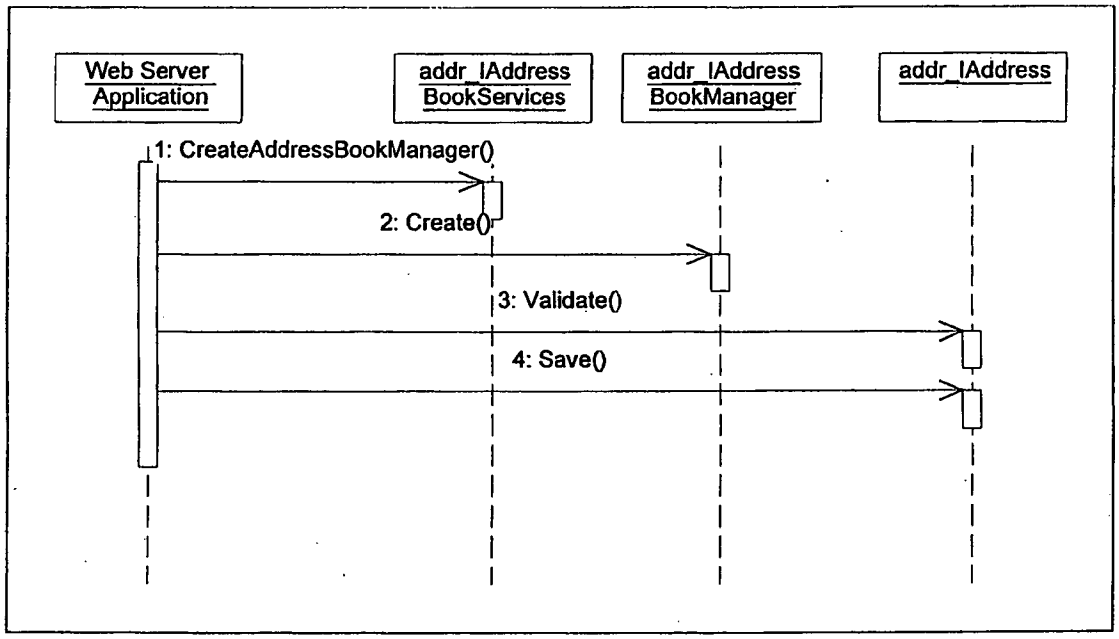


Figure 4. Add a New Recipient Address.

Figure 5. demonstrates how an existing recipient address is deleted from database.

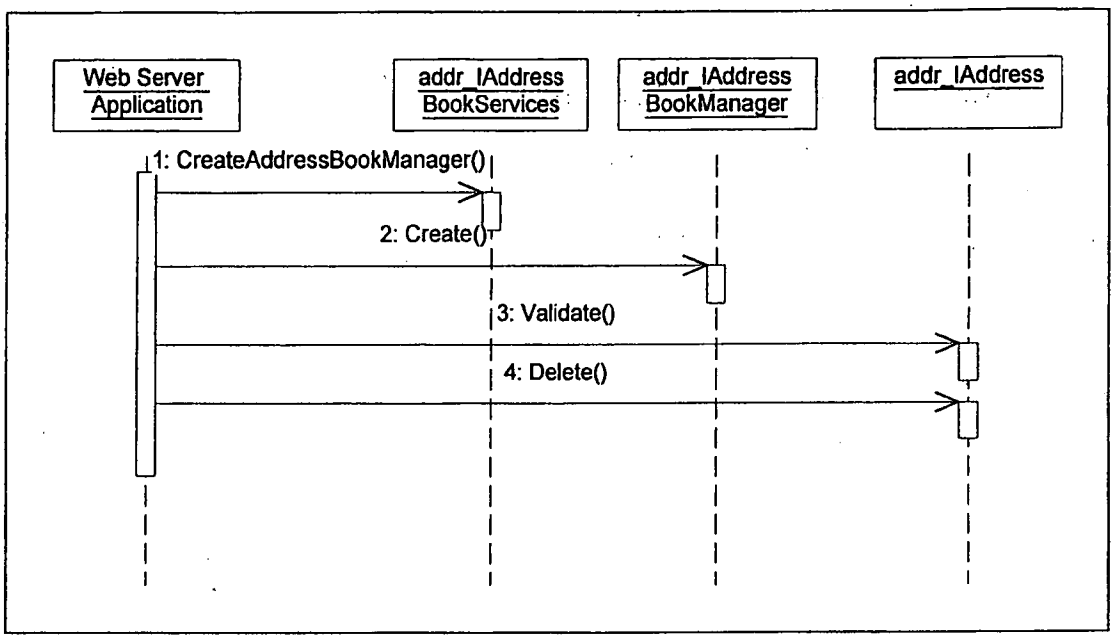


Figure 5. Delete an Existing Recipient Address.

Figure 6. demonstrates how an existing recipient address is updated to the database.

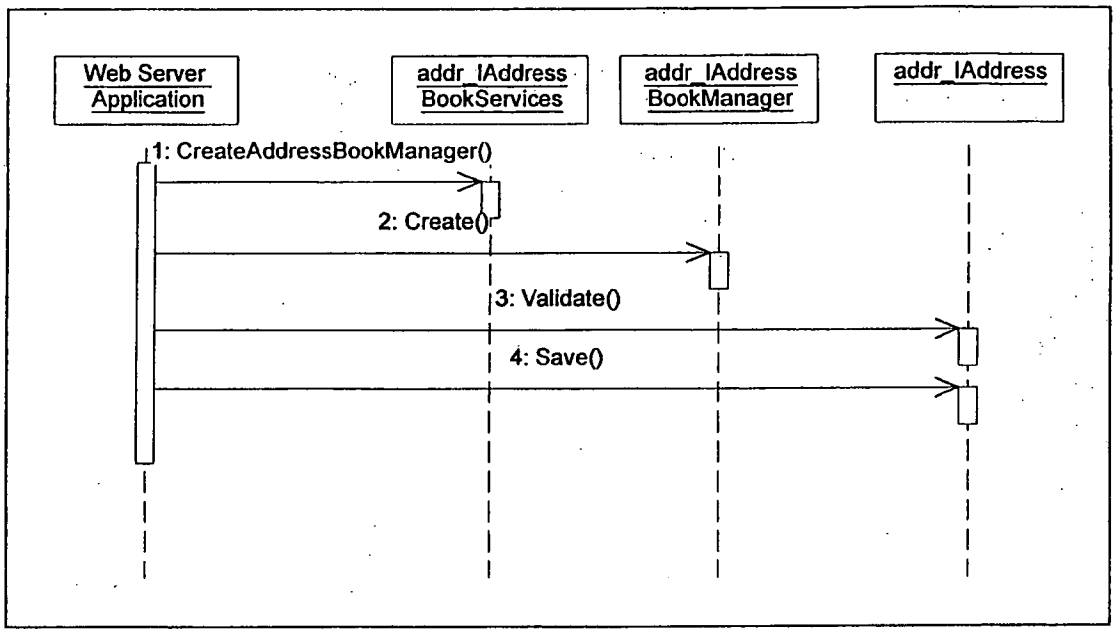


Figure 6. Update an Existing Recipient Address.

EXHIBIT “C”

BEST AVAILABLE COPY

2515 140th Avenue Northeast
Bellevue, WA 98005
Tel: 425.372.1512
Fax: 425.372.1502
steve@moveit.net

MoveIt! Software, Inc.

On-line shipping for everyone, everywhere...

November 26, 1997

Donald B. Mask
President and Chief Executive Officer
College Enterprises, Inc.
21201 Victory Blvd., Suite 270
Canoga Park, CA 91303

RE: LETTER OF INTENT

Dear Don:

The following provides a summary of business terms for the installation and operation of *MoveIt!* Web-based shipping systems at CEI's Pulse Centers and Special Teams campuses.

Parties

College Enterprises Inc. ("CEI")
Attn: Donald B. Mask, President & CEO
21201 Victory Blvd., Suite 270
Canoga Park, CA 91303
Tel: 818-615-0560

MoveIt! Software, Inc. ("MoveIt!")
Attn: Stephen M. Teglovic, President & CEO
2515 140th Avenue Northeast
Bellevue, WA 98005
Tel: 425.372.1512

Business Proposition

CEI and *MoveIt!* will enter into a business alliance whereby *MoveIt!* will install Web-based shipping stations ("Stations") in Pulse Copy & Technology Centers ("Pulse Centers") and on Special Teams campuses. This process will be undertaken in three phases:

- **Phase One:** develop Stations, launch *MoveIt!* Web Client and Network Operations Center (NOC) for installation in Pulse Centers to serve students/low-volume shippers;
- **Phase Two:** integrate CEI's Special Teams debit cards to increase convenience of the payment process; and
- **Phase Three:** expand to include university departmental business.

Term

This business alliance will have an initial term of five years, subject to earlier termination by either party for breach by the other.

Exclusivity

Subject to the terms contained herein, CEI and *MoveIt!* will work exclusively with each other on offering shipping services on US college campuses listed in a schedule to be attached to the definitive agreement.

Revenue Sharing

CEI and *MoveIt!* agree to share 50/50 in the "gross profits" generated by package shipments that are the result of shippers using the *MoveIt!* Stations at Pulse Centers or elsewhere on colleges and universities represented by CEI. For purposes of this letter of intent, "gross profits" shall be defined simply as the difference between the

BEST AVAILABLE COPY

CEI customer charge (i.e. the shipper) and the actual shipping cost charged by the

third-party carrier. Each party shall be responsible for its own operating expenses relating to the Stations and *MoveIt!* shipping system. *MoveIt!* will not charge CEI or the Pulse Centers for the development costs of the technology or for the operation of the *MoveIt!* NOC.

Obligations of the Parties

CEI will own/lease, operate and maintain the Stations at its expense. *MoveIt!* will provide technology, systems set-up and coordination, training and customer support for the Stations at its expense. CEI and *MoveIt!* will work together diligently to define the requirements, marketing plan and deployment plan for the three phases defined above.

Intellectual Property

MoveIt! will own all intellectual property developed by it relating to the Stations and will provide CEI with a nonexclusive license (subject to the exclusivity provision above) to use such intellectual property in connection with the operation of the Stations in Pulse Centers.

Schedule

MoveIt! agrees to install a Beta version (the "Beta") of the Station at a selected Pulse Center for the Phase One customer group approximately five to six months after execution of the definitive agreement. Said Beta will be in operation between 30 and 90 days based on acceptance criteria spelled out in the definitive agreement.

Conditions

CEI agrees to deploy the Stations promptly in each new Pulse Center it opens during the term of the definitive agreement.

Letter of Intent

The Parties understand and agree that these summary terms are non-binding and will not be effective until a definitive agreement is executed by the Parties and ratified by the Parties' respective Boards of Directors.

Don, we're truly excited about this business alliance and appreciate in advance your and your colleagues' help during the development process and beta testing. Would you kindly acknowledge your agreement with the terms herein by signing in the space provided below and I will instruct our attorney to begin drafting a definitive agreement. We look forward to a long and rewarding partnership with you and CEI.

Sincerely,

Stephen M. Teglovic
President & CEO

Acknowledged and agreed on December 6, 1997

By: _____
Donald B. Mask, President & CEO
College Enterprises, Inc.

SMT/ms

cc: *MoveIt! Software, Inc.* Board of Directors

EXHIBIT “D”

EXHIBIT "D"

6. Web Site Overview

Most web sites found on the Internet are collections of organized web pages which, although organized, can be traversed in an unordered fashion. These informational web sites can be discovered through multiple links and their pages bookmarked for later retrieval. The average web user has grown accustom to waiting for graphics and content to download over their limited internet connection, for all web sites are brought to the web user's browser through the same small pipe. The MoveIt! web site is different from the traditional web site, for it provides more than just an information-based collection of web pages. It provides a set of services and is compared more to an application that is running on the web user's machine. Applications have a whole different set of performance requirements in order to be perceived acceptable to the user. Because of this comparison to an application, we have had to mimic the flow that is associated with an application while working with the limitations imposed by a web browser's abilities and connections across the internet.

The web applications, which the MoveIt! web site provides, have had to overcome the unique set of challenges are imposed when designing a tool perceived as both an application and as a web site. Some of the solutions we implemented are:

- Keeping frequently used information in a web page cache and web browser cookies.
- Maintaining an application's state with client cached information.
- Providing secure information transactions with secure socket layers and session keys.
- Using pop-up browser windows to maintain an application's state while performing subtasks.
- Validating forms on the client side to stop simple errors from requiring a server round trip.

Because of the nature of a web site, the design needs to be both flexible and extendible. Some of the ideas we have incorporated into our design are:

- Stateless program flow to allow for interaction with multiple web servers.
- Code reuse with shared set of commonly used JavaScript functions and HTML.
- Data driven screen forms

6.1 Web Site Performance

Speed is the measure of performance for which the MoveIt! web site must optimize. Due to this requirement, the web site applications use a number of techniques to minimize the number of client web browser and web server transactions. One of the slowest processes of the web site is the transferring of information between web browser client and web server. Every time a web user clicks on a hypertext link, a round trip of information to the server must take place for the retrieval of content for the next web page. The following areas explain methods of improving the web site performance.

6.1.1 Client Form Validation

When a user puts information into a form on a web page, they will need to submit it back to the web server to have their data validated and processed. To alleviate some unnecessary round trips, simple validation can be performed on the client browser through JavaScript. The types of validation that can be performed on the client include:

- Verifying there is content in required information fields
- Checking for the "@" symbol in an email address.
- Preprocessing alpha or numeric only separation.
- Required number of password characters
- Changing password content confirmation

If these simple checks find invalid or missing data, they can immediately be brought to the attention of the web user rather than having their form make the round trip only to find that a simple mistake.

The process for performing the client validation involves JavaScript, which is called in one of several methods. The JavaScript can be called when a submit form button is clicked or from a number of different events like OnBlur, which triggers when the current field loses focus. An example of validation is the confirmation of a user's password. Passwords are entered into inputs of type

"password", which have the added security measure of echoing typed characters as asterisks. Below is some sample HTML for a password input, a confirmation password input, and a submit button.

```
<FORM NAME="Test">
Password: <INPUT TYPE=PASSWORD NAME="Password1" SIZE=15> <BR>
Confirm: <INPUT TYPE=PASSWORD NAME="Password2" SIZE=15> <BR>
<INPUT TYPE=BUTTON VALUE="SUBMIT" OnClick="Confirm()">
</FORM>
```

The button input has an OnClick event that is set to call the function Confirm. The corresponding JavaScript verifies that the information matches and alerts the user if there is an inconsistency. Otherwise, the form is sent to the web server.

```
<SCRIPT LANGUAGE="JavaScript">
<!--
   function Confirm()
   {
      if( Test.Password1 != Test.Password2 )
      {
         alert( "Your password and confirmation do not match." );
      }
      else
      {
         Test.submit();
      }
   }
//-->
</SCRIPT>
```

Through JavaScript, a number of similar operations can be used to prevent server round trips on simple mistakes.

6.1.2 Client Information Caching

Once information has been validated and determined acceptable or it has been sent from the server once, it is desirable to not transfer the information every time it is required for display. Some data is stored in the client web browser, hidden from view. This "cached" data can be stored and retrieved through client JavaScript. For example, once a user has logged on to the MoveIt! system, their browser is passed a Session Key. The next transaction the web user makes will include their Session Key, which was held in their cache page. A new Session Key will be generated on the server and sent with the next page the client web browser receives. Another example of caching client information is the storing of data which is being accumulated over several pages and submitted at once for validation.

6.1.3 Selective Secure Transfers

When information needs to be sent between the client web browser and the web server and the information needs to be kept secure, the web page is through a secure socket. To make a socket secure, all messages sent through it are encrypted and then decrypted upon their arrival. Unfortunately, the process used to encrypt and decrypt information involves manipulating the data stream with complicated math functions and large numbers. This process, although very secure, is very slow. To alleviate the process for the web user, only selective information is transferred with a secure socket.

A large amount of the information which is contained within the web site is considered customer non-specific because most customers view it at some point. By keeping only customer specific information secure, the bulk of the data transfers are not penalized by security. One way this is accomplished is that frames that do not contain content that needs to be secure are not passed through a secure socket. Other broadly viewed objects in HTML web pages are graphics, which are referenced and transferred individually. Most graphics do not need to be kept secure and do not need to be sent through secure sockets. Because graphics are comprised of many more bytes than the

EXHIBIT “E”

	by User	
9.1.2	Figure 6. Recipient Report by User	Recipient company name
9.1.3	Figure 6. Recipient Report by User	Recipient address
9.2.1	Figure 6. Recipient Report by User	Package reference No. 1
9.2.2	Figure 6. Recipient Report by User	MoveIt! tracking number
9.2.3	Figure 6. Recipient Report by User	Service
9.2.4	Figure 6. Recipient Report by User	Service options
9.2.5	Figure 6. Recipient Report by User	Carrier
9.2.6	Figure 6. Recipient Report by User	Package status
9.2.7	Figure 6. Recipient Report by User	Ship date
9.2.8	Figure 6. Recipient Report by User	Total package charge
9.3.1	Figure 6. Recipient Report by User	Number of packages for each group
9.3.2	Figure 6. Recipient Report by User	Total charges for each group
9.3.3	Figure 6. Recipient Report by User	Total number of packages
9.3.4	Figure 6. Recipient Report by User	Total package charges
10	Figure 6. Recipient Report	Select custom reports

	by User	

Table 4.Reporting Application Requirements Traceability Matrix

8.1.2 Shipping Application

8.1.2.1 Overview

The Web Shipping Application provides the means by which the necessary information for preparing a package for shipping at a local drop-off site is entered. The package information gathered by the application consists of

- The destination address of the package
- The drop-off site
- The package weight and dimensions
- The carrier, service and service options
- Miscellaneous notes and reference information

This information is gathered in two phases – the "Submit" phase and the "Ship" phase.

During the "Submit" phase a data entry form is displayed for entering the destination address, the drop-off site, the package weight, dimensions, and miscellaneous package notes and reference information. A blue label next to an entry area indicates a required field. Specific instructions describing what must be entered for any field – required or optional – is obtained by simply clicking on the label, and the instructions appear in an information window. Once all the required information is entered, the information is submitted by selecting the "Submit" button. At this point the information is validated for completeness and correctness. If any items are invalid, the labels of the invalid fields are turned red, and the focus is set to the first invalid, data entry field. Clicking on a corresponding red label brings up an information window describing the error and what needs to be done to correct the error. After the invalid fields are corrected, the information is re-submitted by selecting the "Submit" button. This cycle continues until all the required data is correct and complete. When the "submitted" is valid, the "ship" phase commences.

During the "ship" phase a summary of the entered information is displayed along with a rate matrix and a service options selection area. Each cell of the rate matrix displays the rates for shipping the package via carrier / service combination. Initially, these cells either contain the base rates for shipping the package for each carrier and service or "NA" (Non-Applicable) if the service is not available for a carrier or a carrier does not support the service for the destination of the package. The service option area contains checkboxes for selecting one or more service options and associated data entry fields for the service options. A blue label distinguishes required service option entry fields from optional fields. Specific instructions for each service option, data entry field is obtained by clicking on the label of the data entry field, and the instructions appear in an information window. As one or more service options are selected, the rates in the rate matrix are adjusted to include the selected service option charges. If a service does not support a selected service option, the rate displayed in the cell changes to "NA".

To "ship" a package a service must be selected. The service to be used to ship the package is selected by clicking any cell in the rate matrix that does not have "NA". Once a service is selected, selecting the "Ship" button will ship the package. At this point the service and service option information is validated. Any invalid entries will most likely be due incomplete or incorrect service option supporting information. The labels of the invalid items are turned red and the focus is set to the first invalid field. As in the "submit" phase, clicking on the label of an invalid field brings up an information window describing the error and what to do to correct the error. After correcting the invalid fields the, package is re-shipped by selecting the "Submit" button. This cycle is repeated until all the required fields are valid.

Once the package has been shipped, the Web Shipping Application enters the final, non-data entry phase. During this phase a final summary is displayed showing the package information displayed earlier during the start of the "Ship" with the addition of the selected carrier, service, service option(s)

and associated data, an application generated tracking number, and up to five rates for the package. The five rates represent the rates for the package weights that bracket and include the billable weight of the package. If the shipped package is a letter only one rate is displayed.

In each of the three phases of the Web Shipping Application several buttons are displayed along with the other phase related information. These buttons are the means by which the Web Shipping Application navigates the web pages that comprise the application. During the "Submit" phase three buttons are displayed:

- A "Locator" button for selecting the drop-off site of the package. A separate window is brought up for selecting the drop-off site.
- A "Cancel" button for resetting the package information on the "Submit" web page back to the system defaults and user preferences.
- A "Submit" for submitting package information for validation.

During the "Ship" phase three button are also displayed:

- An "Edit" button to edit the current package information.
- A "Cancel" button for resetting the package information entered on the "Submit" web page back to the system defaults and user preferences
- A "Ship" button to store the package information

During the last phase two buttons are displayed along with the final summary information

- A "Ship Another Package" button for shipping another package.
- A "Done" button for exiting the Web Shipping Application. This button is a duplicate of the "Done" navigation button and is provided as a convenience on this last page.

8.1.2.2 Structure and Identification

Attribute	Value
Prefix	ship
Directory Name	Shipping

Table 5. Shipping Structure and Identification

8.1.2.3 User Interface Design

8.1.2.4 Transacti n Sets

This table below describes the transactions that the Web Shipping Application will have with the Web Server.

Transaction Sets	
Browser Request	**Server Response(s)**
Submit package information	Validates the package information. If successful, a rate matrix is returned. Otherwise, the list of invalid package fields and corresponding error and correction text is returned.
Ship a package	Validates the package information. If successful, a set of rates bracketing and including the billable weight rate is returned. Otherwise, the list of invalid package fields and corresponding error and correction text is returned.
Edit a package	The current package information is retrieved from the server and the initial Shipping data entry page is populated with the package data.

Table 6. Shipping Transaction Sets

8.1.2.5 External Dependencies

The physical dependencies that the Web Shipping Application has are:

Type	Name	Description
Utility Package Classes		
COM Object	ship_IShipping	Shipping COM object
COM Object	ship_IPackage	Package COM object
COM Object	ship_IRateInfo	Rating Information COM object
COM Object	ship_IServices	Service and service option COM object
COM Object	IDictionary	Microsoft dictionary
DLL		
Static Library		

Table 7. Shipping External Dependencies

8.1.2.6 Requirements Traceability Matrix

This table lists all the individual line item requirements met by the Shipping application

Requirement ID	Figure	Comments

EXHIBIT “F”

9.3.8 Database Servic s

9.3.8.1 Overview

The database services will provide Wolverine sever components with database connection handling methods.

Because of the database-distributed architecture, server components will connect to different databases on different servers. The database services will make sure each component connects to the right server and the right database, as well as implementing a failure recovery scheme if a connection fails.

9.3.8.2 Structure and Identification

Table 1.1.1-1 defines the naming prefix used to define objects within the component and it defines the name of the component.

Attribute	Value
Prefix	dbs
Name	database
Type	DLL

Table 20. Database Services Structure and Identification.

9.3.8.3 Exported Interfaces

9.3.8.3.1 Classes

This section defines the classes that are exported from the Reporting service. Each class is defined in a table that describes its methods and properties.

dbs_TDatabaseServices

The methods of the dbs_TDatabaseServices object take a secur_IUser interface pointer and provide access to the dbs_TDatabaseAdmin objects.

The table below shows methods and properties of dbs_TDatabaseServices.

Class: dbs_TDatabaseServices	

Constructors/Destructors	
dbs_TDatabaseServices	Default constructor that initializes class data members. Copy constructor is private and not implemented.
~ dbs_TDatabaseServices	
Methods	
CreateDatabaseConnection()	Returns a database connection interface pointer.
Properties	
<None are required>	
Operators	
<None are required>	

Table 21. Class dbs_TDatabaseServices Descriptio

dbs_TConnection

The dbs_TConnection prepares the SQL statements. It also exposes the dbs_TStatement object, which allows execution of statements. The table below shows methods and properties of dbs_TConnection.

Class: dbs_TConnection	
Constructors/Destructors	
dbs_TConnection	Constructor takes connection handle and assigns it to member data that holds this value. Default and Copy constructor is private and not implemented.
~ dbs_TConnection	
Methods	
CreateDatabaseStatement	Returns a database statement interface pointer.
GetConnection	Allocate a connection handle and returns a connection interface pointer.
Properties	
<None are required>	
Operators	
<None are required>	

Table 22. dbs_TConnection Description.

dbs_TStatement

The dbs_TStatement executes the SQL statements. The table below shows methods and properties of dbs_TStatement.

Class: dbs_TStatement	
Constructors/Destructors	
dbs_TStatement	Default Copy constructor initialize all member pointers to Nulls.
~ dbs_TStatement	Free all pointers allocated
Methods	
_Init	Restricted method that initializes an allocated statement handle.
FillMap	Restricted method for saving results in a map
CleanUp	Restricted method for releasing resources
BindColumns	Restricted method for binding columns used in ODBC SQLFetch
Execute()	Executes the statement. It will bind parameters if there are any and fetch only the FIRST record set if it's a select statement.
Prepare	Takes the SQL statement and allocate a statement handle. It then prepares the statement. And it returns a statement interface pointer. It is also passed flag that indicates if the statement is a select statement or not.
GetNext	Used only after executing a select statement. Execute will return the first row of the result set. Next will get the next record.
GetFirst	Move to first record
GetLast	Helper function for navigation. Used only after executing a select statement. It returns the last row.
MoveAbsolute	Moves to n record if exists
MoveRelative	Moves n records relative to current position
GetPrevious	Returns previous record from current position
IsBOF	Helper function for navigation. Used to determine the beginning of a result set
IsEOF	Helper function for navigation. Used to determine the end of a result set, will call before Next() is called to determine if there is a next record
Properties	
<None are required>	
Operators	

<None are required>	

Table 23. dbs_TStatement Description.

9.3.8.3.2 Free Functions

Free functions are defined in the table below. Free functions are functions not associated with a particular class.

Free Functions	Description
<None are required>	

Table 24. Free Function Descriptions

9.3.8.3.3 COM Interfaces

This section defines the COM interfaces that are exported from the database services. Each interface is defined in a table that describes its methods and properties.

COM Class dbs_DatabaseServices

The COM Class DatabaseServices provides connection and prepare statements for other server components. It implements the following interfaces that are defined in the following table

Attribute	Description
ProgID	MSI.dbs_DatabaseServices
Supported Interfaces	dbs_IDatabaseServices IDispatch IObjectControl ISupportErrorInfo IUnknown

Table 25. COM Class DatabaseServices

COM Class dbs_Connection

The COM Class dbs_Connection prepare statements for other server components. It implements the following interfaces that are defined in the following table

Attribute	Description
ProgID	MSI.dbs_Connection

Supported Interfaces	dbs_IConnection IDispatch IObjectControl ISupportErrorInfo IUnknown

Table 26. COM Class dbs_DatabaseConnection

COM Class dbs_Statement

The COM Class dbs_Statement executes SQL statements for other server components. It implements the following interfaces that are defined in the following table

Attribute	Description
ProgID	MSI.dbs_Statement
Supported Interfaces	dbs_IStatement IDispatch IObjectControl ISupportErrorInfo IUnknown

Table 27. COM Class dbs_Statement

Interface dbs_IDatabaseServices

A high level interface that allows access to the rpt_IDatabaseConnection

The table below shows methods and properties of rpt_IDatabaseServices.

Interface: dbs_IDatabaseServices	
Methods	
CreateDatabaseConnection()	Returns a database connection interface pointer.
Properties	
<None are required>	
Operators	
<None are required>	

Table 28 dbs_IDatabaseServices Interface Description.

dbs_IConnection

The dbs_IConnection prepares the SQL statements. The table below shows methods and properties of dbs_IConnection.

Class: dbs_IConnection	
Methods	
CreateDatabaseStatement	Returns a database statement interface pointer.
GetConnection	Allocate a connection handle and returns a connection interface pointer.
Properties	
<None are required>	
Operators	
<None are required>	

Table 29. dbs_IConnection Description.

Interface dbs_IStatement

The dbs_IStatement object executes SQL statements and verifies data validation before saving them in the database. The table below shows methods and properties of dbs_IStatement.

Class: dbs_IStatement	
Methods	
_Init	Restricted method that initializes an allocated statement handle.
Execute()	Executes the statement. It will bind parameters if there are any and fetch only the FIRST record set if it's a select statement.
Prepare	Takes the SQL statement and allocate a statement handle. It then prepares the statement. And it returns a statement interface pointer.
GetNext	Used only after executing a select statement. Execute will return the first row of the result set. Next will get the next record.
GetFirst	Move to first record

GetLast	It returns the last row.
MoveAbsolute	Moves to n record if exists
MoveRelative	Moves n records relative to current position
GetPrevious	Returns previous record from current position
IsBOF	Used to determine the beginning of a result set
IsEOF	Used to determine the end of a result set.
Properties	
<None are required>	
Operators	
<None are required>	

Table 30. dbs_IStatement interface Description.

9.3.8.3.4 Interaction Diagrams

To better understand the interactions with database Services and other components of project Wolverine, interaction diagrams are presented. Each diagram depicts a scenario showing one or more components interacting. The diagram does not depict what is going on inside of the database services package. It is treated here as a black box. Figure 1.1.1-1 shows how the interaction between a Report Web Application and the dbs_IDatabaseServices, rpt_IReportServices, rpt_IReportManager, rpt_IReportCollection and rpt_IReport interfaces. The diagram depicts a simple error free scenario. In step 1, the web app calls CreateObject("ReportServices"). The ReportServices object will call method CreateReportManager(). Inside the function CreateReportManager(), the following will be called:

1. Create Database Services passing it a secure user
2. dbs_IDatabaseServices::CreateDatabaseConnection()
3. dbs_IDatabaseCopnnection::GetConnection(). This function establishes the connection and prepares statements that can be prepared in advance.
4. Once a connection is established, then the function CreateReportManager() will Create Report Manager object, passing it the connection and statements handles.

The report manager then uses the handles to do its work.

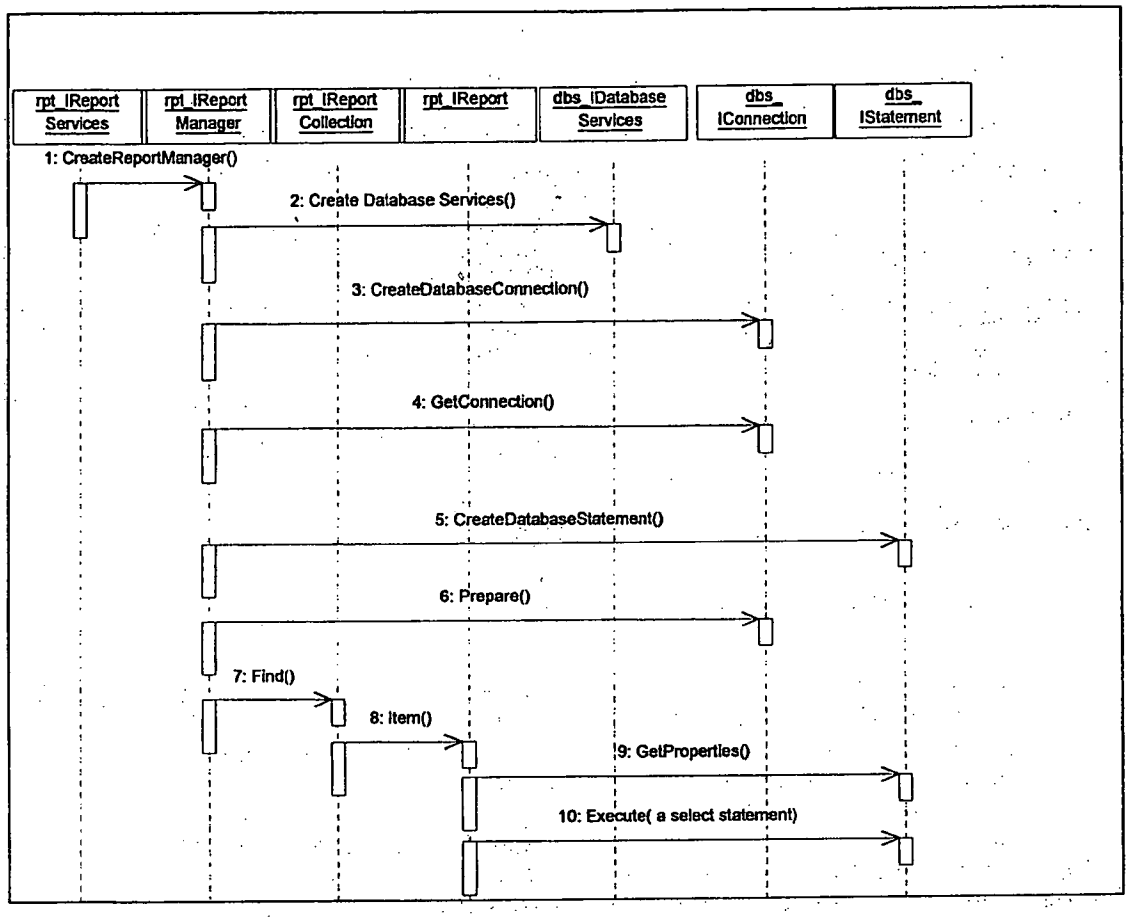


Figure 6. Report Web application and DatabaseServices Scenario Interaction Diagram

9.3.8.3.5 Messages

The table below defines messages that either Database Services publishes or subscribes to.

Name	*Description*
_STATUS.INFO	
STATUS.INFO.DATABASE_INITIALIZATION	Message sent at the start of initialization.
_STATUS.WARNING	
None are required.	
_STATUS.ERROR	
STATUS.ERROR.DATABASE.DB_CONNECTION_FAILURE	Couldn't connect to the database during the initialization process
STATUS.ERROR. DATABASE.DB_PREPARE_FAILURE	Couldn't prepare ODBC statements during initialization process
STATUS.ERROR. DATABASE.DB_INSERT_FAILURE	Couldn't add a new custom report
STATUS.ERROR. DATABASE.DB_UPDATE_FAILURE	Couldn't update a custom report
STATUS.ERROR. DATABASE.DB_DELETE_FAILURE	Couldn't delete a custom report
_JOB	
None are required.	

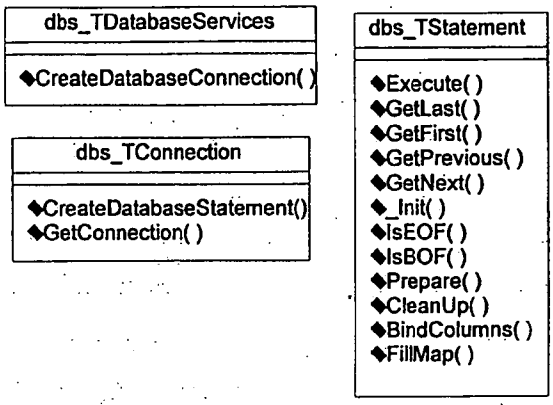
Table 31. Publish and Subscribe Messages Supported by Database Service

9.3.8.4 Implementation Details

9.3.8.4.1 Classes

This section is the same as section "Exported Interfaces/ Classes".

9.3.8.4.2 Class Diagrams



9.3.8.5 External Dependencies

The following table lists dependencies that this package has on components and tables.

Type	Name	Description
Utility Package Classes		
COM Object		
DLL		
Static Library	ATL	Active X Template Library

Table 32. Database Services External Dependencies

EXHIBIT “G”

7. Rate Shop Application

7.1.1.1 Overview

The Rate Shop Application manages the on-screen-rate shopping. The user will be able to enter the following shipping information:

1- Origin Zip Code or Drop off Location

2- Destination Zip Code

3- Expected Drop off Date

4- Weight

5- Packaging type

6- Additional Handling

Once the above information is submitted, the MoveIt! Server components will validate the submitted shipping information; if the validation is successful, the Rating page will be displayed otherwise a descriptive error will be displayed.

7.1.1.2 Structure and Identification

Attribute	Value
Prefix	rate
Directory Name	Rateshop

Table 1. Rate Shop Structure and Identification

7.1.1.3 User Interface Design

The Rate Shop application entry page is the Rate Shop page.

The user enters here the necessary shipping information to get a rate for the package.

The user will be able:

1- To enter either an origin zip code or to select a location from a list of available drop off locations.

2- To enter destination zip code

3- To select expected drop off date from a list of the next seven working days

4- To select packaging type from a list of packaging types

If "Your Packing" selection is selected, then the user must enter:

- Height
- Width
- Length

5- To select Additional Handling

Figure 1 shows the Rate Shop page.

BEST AVAILABLE COPY

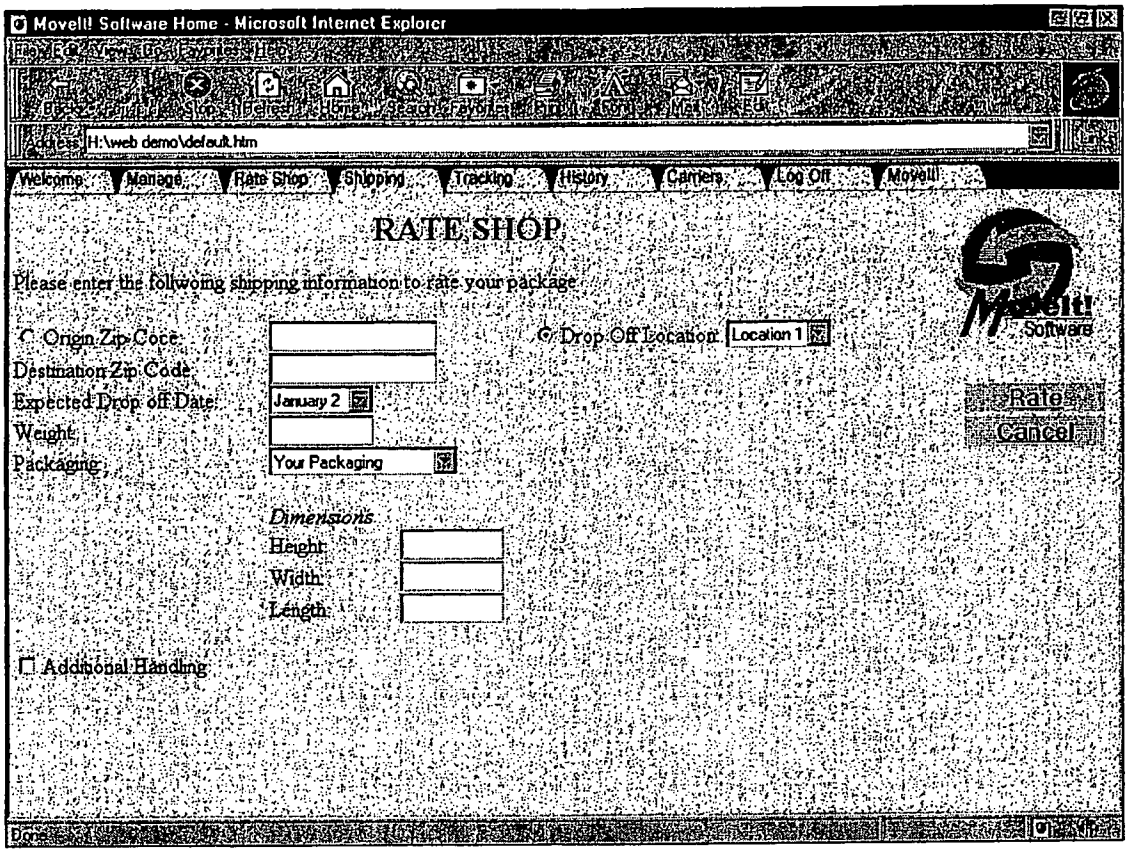


Figure 1. Rate Shop page

Once the information in the Rate Shop page are submitted and validated by MoveIt! Server components, the Rating page will be displayed.

The Rating page displays the following:

1. The expected ship date
2. Service options controls
    - Declared value
    - Outbound alert
    - Priority delivery notification
    - Verbal confirmation of delivery
    - Saturday delivery
3. Rate grid. It displays valid rates and services for submitted data and for each carrier
4. A condensed account agreement
5. Package weight

BEST AVAILABLE COPY

Figures 2. Shows a screen shot of the Rating page

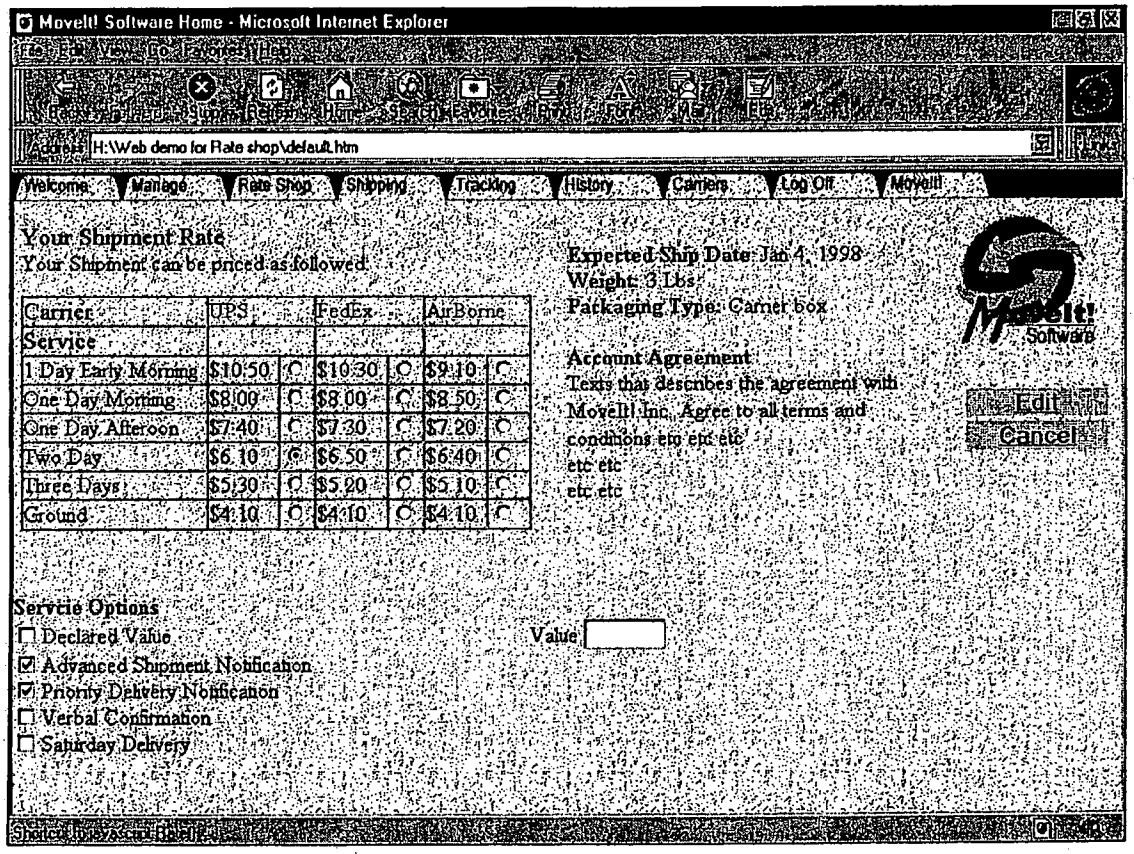


Figure 2. Rating page

7.1.1.4 Data Flow Diagrams

The Data Flow diagram shows how the user accesses the Web Rate Shop Application, or the Rate Shop State "rate_Main", from the welcome screen by selecting the Rate Shop selection.

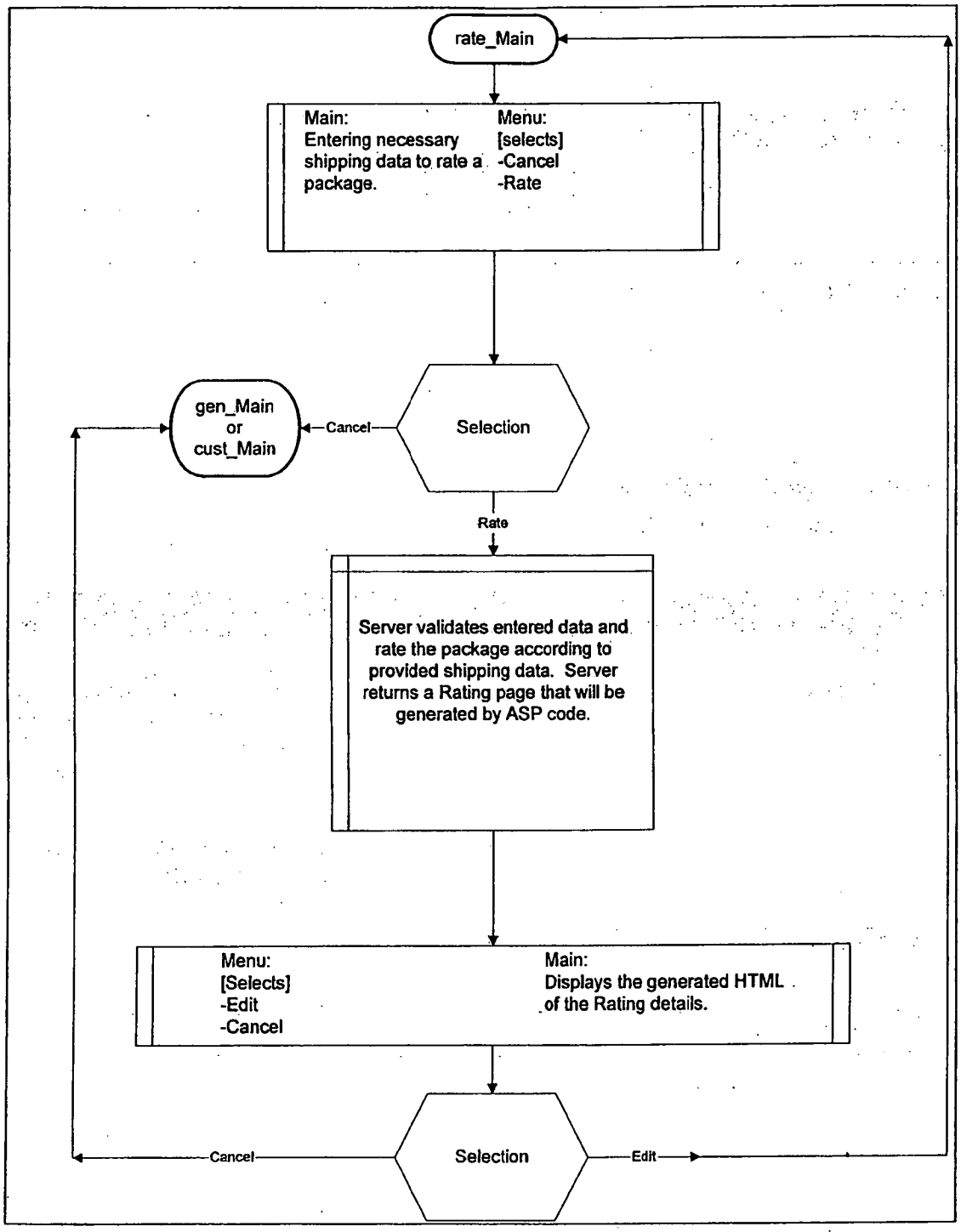


Figure 3. Rate Shop data flow diagram

7.1.1.5 Transaction Sets

This table describes the transactions that the Web Reporting Application will have with the Web Server.

Transaction Sets	
Browser Request	**Server Response(s)**
Rate	Submit user's necessary shipping data to rate a package and return a HTML rating page

Table 2. Reporting Transaction Sets

7.1.1.6 External Dependencies

The physical dependencies that the Web Reporting Application has are:

Type	Name	Description
Utility Package Classes	NA	
COM Object	rate_IRating biz_IBusinessRules	Uses the carrier, service, drop-off site and destination information, weight, and dimensions to calculate a matrix of rate information per carrier and per service. The main function of this service will be to provide validation and calculation of small parcel packages.
Database Tables		Rate tables
Static Library	NA	

Table 3. Rate shop External Dependencies

EXHIBIT “H”

8.1.3 Tracking

8.1.3.1 Overview

The tracking application allows the user to gather information about a package that has been previously shipped with the MoveIt! shipping system. Pertinent information such as delivery status, location, and recipient will be returned if available.

The Tracking Web Application is designed to allow the user to:

- Track packages shipped through MoveIt! with a MoveIt! supplied tracking number.
- Track packages that may or may not have been shipped through MoveIt! with a carrier supplied tracking number.
- Track packages from a historical list of packages shipped through MoveIt!.

The Design is intended to be flexible and easily maintained based on the dynamic nature of a Web site. The following specific things were kept in mind when designing the Tracking application:

- Flexible carrier support.
- Support for variable amounts and types of information provided by carriers.
- Pass control to history when tracking from history is desired.

8.1.3.2 Structure and Identification

The table below defines the naming prefix used to define objects within the application and it defines the directory name where all the components of the application will reside when under configuration control.

Attribute	*Value*
Prefix	track
Directory Name	Tracking

Table 8.1.3-1 Tracking Structure and Identification.

8.1.3.3 User Interface Design

The following two screenshots illustrate a typical tracking session.

BEST AVAILABLE COPY

8.1.3.3.1 Tracking Number Entry User Interface Design

illustrates the initial entry to the tracking application. Here the user is prompted to enter a tracking number and, if it is not a MoveIt! tracking number, what carrier the tracking number belongs to.

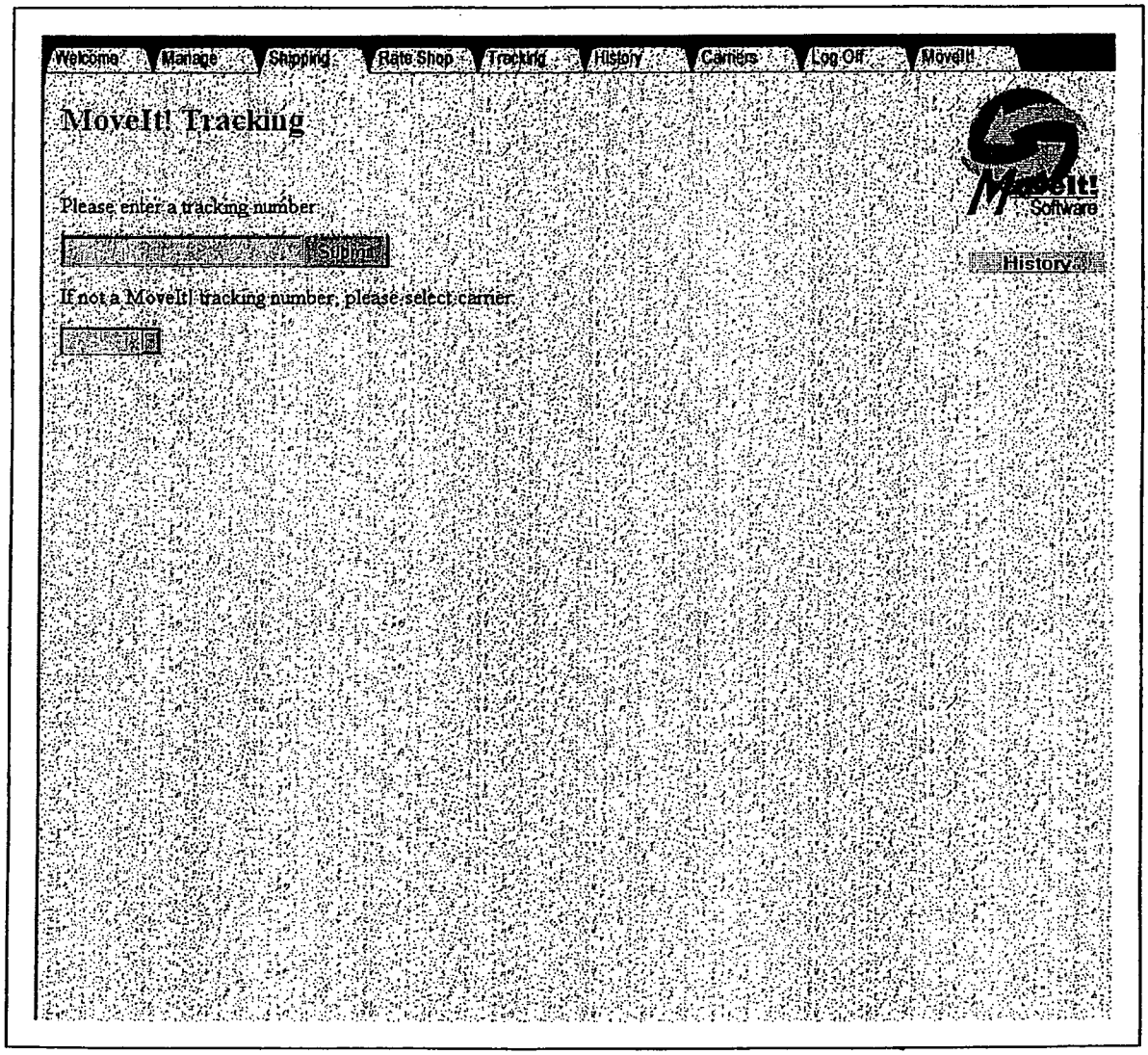


Figure 13. Main Tracking screen

8.1.3.3.2 Tracking Result User Interface Design

The following diagram represents a successful track request using a MoveIt! tracking number. Note that the user is allowed to immediately track another package from the detail screen.

BEST AVAILABLE COPY

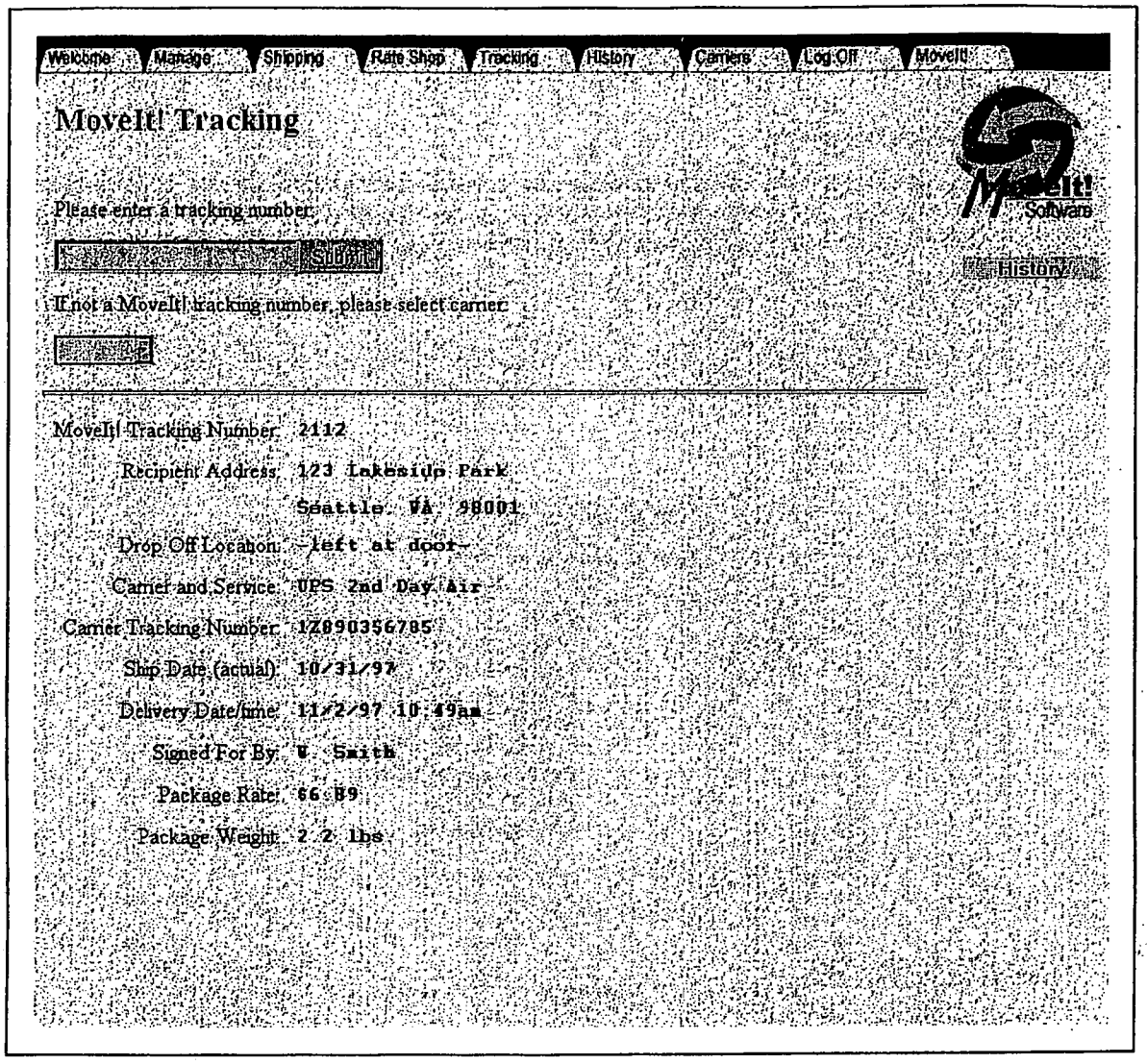
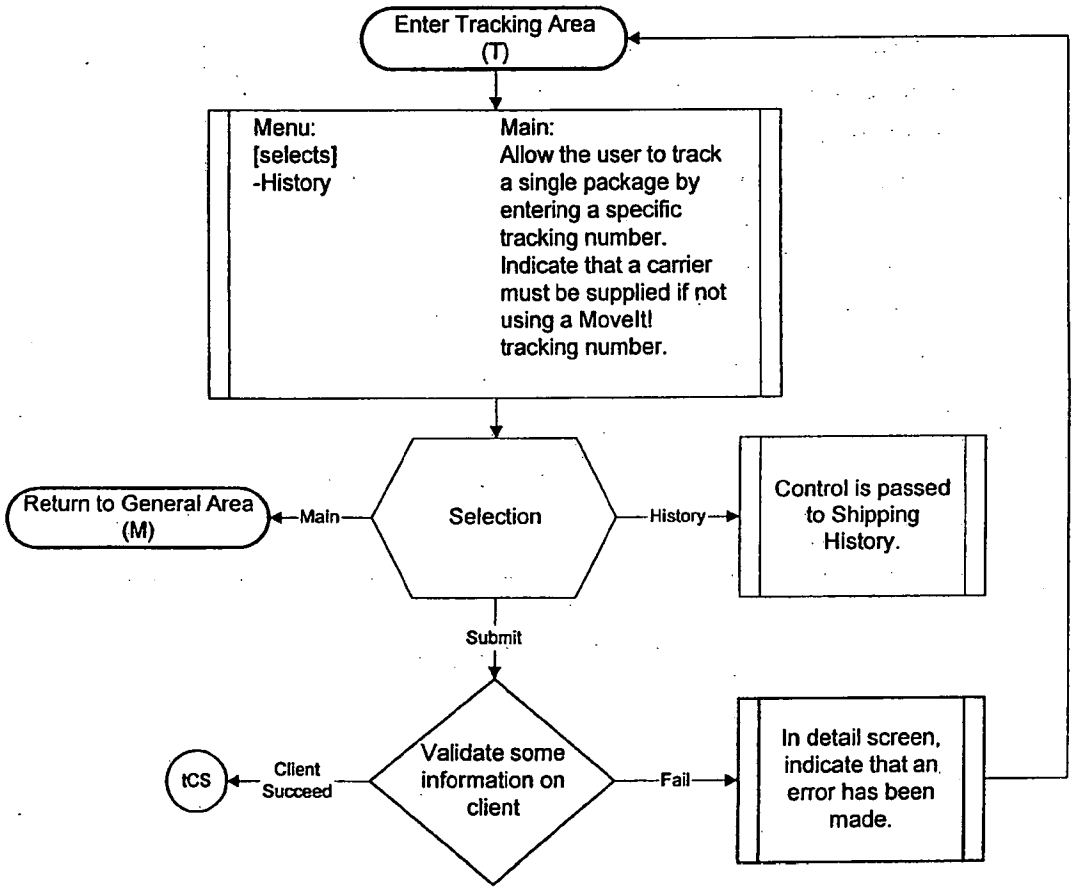
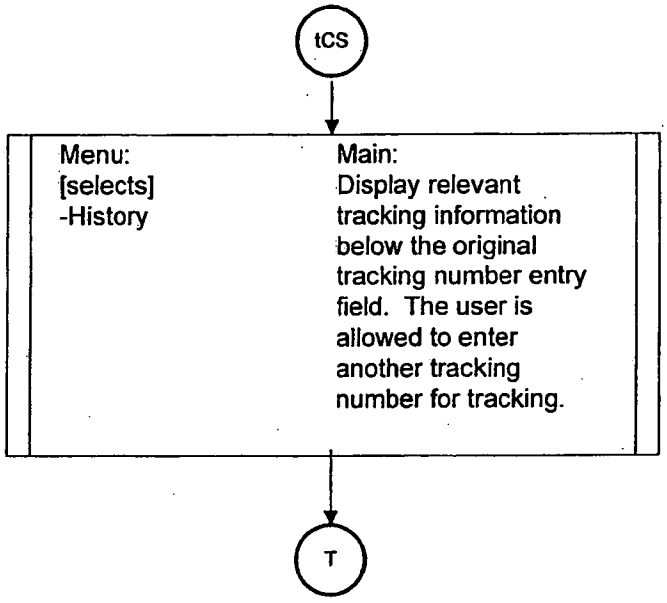


Figure 14. Tracking detail screen

8.1.3.4 Data Flow Diagrams

The following figure details what happens when a user enters the tracking application and submits a tracking request. Note that when an error is made, the flow is similar to a successful tracking request except that the detail screen will contain error information instead of tracking detail.





8.1.3.5 Transaction Sets

The table below (Table 6.5.2-1) describes the transactions that the Web Browser will have with the Web Server.

Transaction Sets	
Browser Request	**Server Response(s)**
Submit Track Request	Current tracking information for the entered tracking number.
Track from History	Control is passed to the Shipping History view

Table 8.1.3-2. Tracking Transaction Sets

8.1.3.6 External Dependencies

The table below lists dependencies that this application has on other components and database tables.

Type	Name	Description
Database Tables	Shipment	

COM Objects	IDictionary	Microsoft dictionary
COM Objects	track_ITrackingServices	Tracking services
COM Objects	track_ITrack	Tracking interface
COM Objects	track_ITrackingInfo	Tracking details interface

Table 8.1.3-3. Tracking External Dependencies

EXHIBIT "I"

	method is called.
IsDirty()	Returns a boolean value indicating whether any property has been modified since the object was created.
IsValid()	Returns a boolean value indicating whether a successful validation had been performed since the last property change. Does not call the Validate() method.
IsReadOnly()	Returns a boolean value indicating whether this is a read-only interface. If it is, the following methods should return the BIZ_READONLY_ACCESS_ALLOWED hresult: PutProperty(), PutProperties(), IsPropDirty(), IsDirty(), IsValid(), and Validate().
IsFromStorage()	Returns a boolean value indicating whether the object was originally retrieved from data storage. This is used when saving an object back into storage to determine if an Insert or Update needs to be performed.
Properties	
OID	The OID (Object Identifier) of the current object. (read-only)
ObjectTag	A human readable name for the object. For example "REPORT". Names shall be in all uppercase. (read-only)
ReadOnly	Value is either 0 or 1. 1 Indicating the object is read-only. If an object is stored in the database as read-only, it will be retrieved initially in a read-only state. (read-only)
<List of all other properties and their descriptions that can be retrieved from the property collection that are specific to the particular business object. These are NOT properties in the COM sense.>	

Table 12. Interface *biz_IBusinessObject*

COM Class Data Dictionary

Attribute	Description
ProgID	MSI.biz_DataDictionary
Supported Interfaces	biz_IDataDictionary IDispatch IObjectControl ISupportErrorInfo IUnknown

Interface biz_IDataDictionary

The data dictionary interface allows for the retrieval of data dictionary items. It also adds a convenient function to validate a value against a particular data dictionary entry.

Interface: biz_IDataDictionary	
Initialization/Un-initialization	
Init()	Loads a data dictionary as specified by the passed in name from persistent storage. Not required to be used if created orginally with biz_IDataDictionary interface.
Methods	
Count()	Based on a passed in filter string, retrieves the number of a DataDictItem objects matching the filter. The count is set to zero(0) if nothing matches.
Create()	Creates an empty thing. The properties are filled with all the property names and any default values are assigned. An OID is generated and the Name property is assigned at this time.
CreateFilterUsingExample()	Uses the properties of the passed in biz_IDataDictItem to build a filter string that is returned. Use the filter string on future calls to methods that require a filter.
Delete()	Deletes a DataDictItem using the passed in OID.
Find()	Based on a filter string, retrieves the first occurrence of a DataDictItem object matching the filter. The returned DataDictItem is NULL if no item matches.
Items()	Returns a util_IMap collection. The number and ordering of DataDictItems in the collection is determined by the passed in filter and the current user. The key field holds OIDs of data dictionary items.
Load()	Based on an OID retrieves a single DataDictItem object.
Save()	Takes a biz_IDataDict as an argument and stores it in data storage. The Validate() method of DataDictItem is called prior to any attempt to store the object. If the object is currently in data storage, it is updated otherwise it is inserted.
ValidateValue()	Takes a VARIANT that represent a possible value for a data item of the current type and the name of a data dictionary item and validates it. Returns both a boolean value and a status code. The status code is used to indicate whether the value was modified as a result of the validation, for example, converting a string to all uppercase letters.
Properties	
None are required.	

Interface biz_IDataDictItem

This is an implementation of the biz_IBusinessObject interface. This interface is only used in conjunction with the Data Dictionary Administrator to create and save data dictionary items. The properties are defined below in the biz_IDataDictItemReader.

Interface biz_IDataDictItemReader

The table below defines the data dictionary item interface. This interface is only used in conjunction with the Data Dictionary Mangager to retrieve data dictionary items for read-only usage.

Interface biz_IDataDictItemReader	Description
Initialization/Un-initialization	
None are required.	
Methods	
GetProperties()	Creates and returns an IDictionary interface of properties. Wildcards are supported that allow filtering of the returned properties.
GetProperty()	Returns the value of a single property as a VARIANT. The property is indexed by a key.
GetPropValidValues()	Takes a property name and returns an IStringList interface of allowed values. Used for example to populate a list box.
ValidateValue()	Takes a VARIANT that represent a possible value for a data item of the current type and validates it. Returns both a boolean value and a status code. The status code can be used to indicate whether the value was modified as a result of the validation, for example, converting a string to all uppercase letters.
Properties	All properties are read-only
Default	VARIANT representing a default value if one applies.
ID	A unique integer identifier that represents this data dictionary item.
Max	Maximum allowed value. For strings this equal to the maximum allowed length (number of characters).
Min	Minimum allowed value. For strings this equal to the minimum allowed length.
Name	The name of this data dictionary item. For example: ADDRESS.POSTAL_CODE.
NullValue	Contains a value that is appropriate with this data dict. item to represent NULL.
ObjectName	Object name. Always set to "DATADICTITEM" (read-only)
OID	The OID (Object Identifier) of the current object. (read-only)
Precision	Number of allowed digits to the left of the decimal point.

Scale	Number of digits to the right of the decimal point.
SQLType	Integer represent the appropriate ODBC SQL type to store data that maps to this data dictionary item.
Traits	Bitmask of traits. The traits are: Uppercase, Lowercase, Mixed, Nullable, DefaultVal, HasSet, StripWhite, StripReturn, Read-only,
ValidValues	Contains a SafeArray of Variants that contain a set of allowed values for this item.
VariantType	One of the VT_ types for data that maps to this data dictionary item.

9.3.5.3.4 Class Diagrams

None are required.

9.3.5.3.5 Interaction Diagrams

The interaction diagram below (Figure 3) shows how a typical Server Component would use the data dictionary object to validate data prior to saving it in the database. The diagram should just one call to Validate(). In most cases this method will be called for each data item that has been modified.

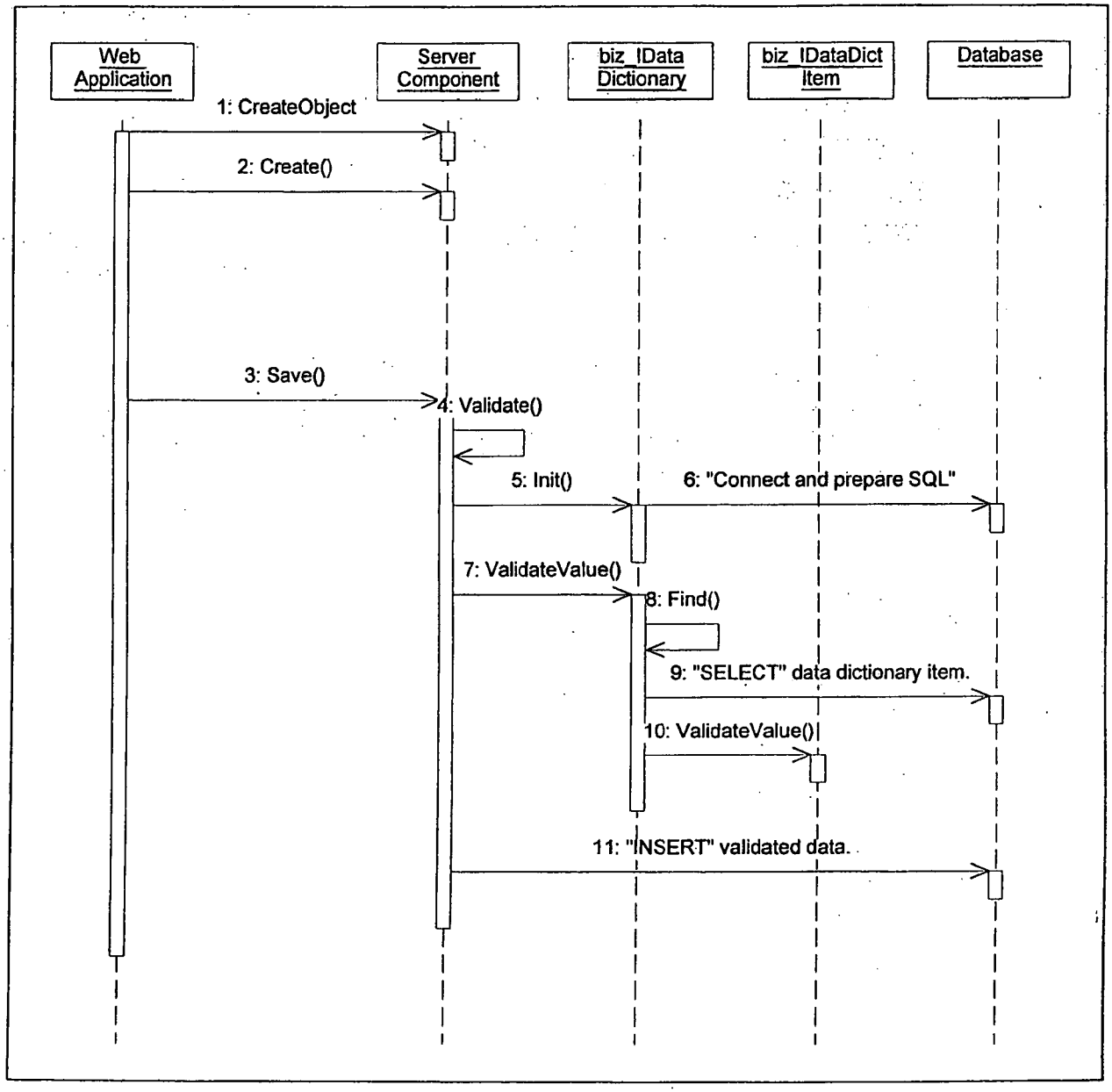


Figure 3. Interaction Diagram showing how the Data Dictionary interfaces can be used.

9.3.5.3.6 Messages

The table below defines messages that either <Gadget> Services publishes or subscribes to. For more information on messages, see the Event Services design section in this document.

Name	Description
_STATUS.INFO	
BUSINESS.DATADICTIONARY_LOADED_SUCCESSFULLY	Message sent whenever a data dictionary is loaded. The name of the dictionary is included as a message parameter.
_STATUS.WARNING	
None are required.	
_STATUS.ERROR	
BUSINESS.DATADICTIONARY_NOT_FOUND	Error message if a data dictionary with the requested name cannot be located.
_JOB	
None are required.	

Table 13. Publish and Subscribe Messages Supported by Business Rule Services.

9.3.5.4 Implementation Details

None are required.

9.3.5.4.1 Classes

None are required.

9.3.5.4.2 Class Diagrams

9.3.5.4.3 Free Functions

None are required.

9.3.5.4.4 Interaction Diagrams

None are required.

9.3.5.5 External Dependencies

There are many database dependencies that will need to be added.

Type	Name	Description
Database Table	DATADICTIONARY	Table that stores all the datadictionary information.
Utility Package Classes	util_Error	
COM Object	CVariantDictionary	Microsoft Variant map class.
DLL		None are required.
Static Library	ATL	ActiveX Template Library

Table 14. Business Rules Service External Dependencies

9.3.5.6 Requirements Traceability Matrix

The table below traces functional requirements to this design section.

Requirement ID	Class/Interface	Comments

Table 15. Business Rule Services Requirements Traceability Matrix

9.3.6 Carrier Connectivity Services

9.3.6.1 Overview

The purpose of the Connectivity service is to allow a standard interface for communicating with various carriers electronically. Each carrier will have its unique method for sending electronic data for example UPS currently uses a custom protocol called UPSLink/Z for which ups provides a communications DLL. Since each carrier will have its own form of communications the connectivity service will provide a single interface that can be used to send electronic data to any of the carriers.

9.3.6.2 Structure and Identification

The table below defines the naming prefix used to define objects within the component and it defines the name of the component.

Attribute	*Value*
Prefix	conn
Name	Carrier Connectivity
Type	DLL

Table 9.3.6-1. Carrier Connectivity Structure and Identification.

9.3.6.3 Exported Interfaces

Exported interfaces are all classes, free functions, & COM interfaces that other packages may need to use. In other words, these are the services that this package is providing. Function arguments are described but not shown. This is done to make the document more readable and maintainable as the design evolves.

9.3.6.3.1 Classes

None are required.

9.3.6.3.2 Free Functions

None are required.

9.3.6.3.3 COM Interfaces

This section defines the COM interfaces that are exported from the Carrier Connectivity. Each interface is defined in a table that describes its methods and properties.

Interface conn_ICarrierConnect

A description of the interfaces roles and responsibilities goes here.

Interface: conn_ICarrierConnect	
Identifier	
ProgID	MSI.conn_ICarrierConnect
Initialization/Un-Initialization	
None are required.	
Methods	
GetDirectTrackingInterface()	This method will provide an interface pointer that can used for package tracking through a direct connection to the shipper. Parameters: An enumerated constant that represents that carrier that the package should be tracked through. A pointer to an conn_IDirectTrack interface pointer.
GetWebTrackingInterface()	This method will provide an Interface that can be used for package tracking through the Web sight of the carrier. Parameters: A pointer to a conn_IWebTrack interface pointer.
GetRegistrationParameters()	This method will be used to get a collect of all parameters that are required to register a shipper with a give carrier. The caller will then be able to enumerate over the collection and fill in all of the data that is required by the carrier to register a new shipper. Parameters: An enumerated constant that represents the carrier that the shipper will be registered with.

	A pointer to an IMap interface that will be filled in with the parameters that are required to register the shipper.
RegisterShipper()	This method will be used to register a shipper with a given carrier. This function is synchronous and will return until the shipper has either been registered successfully or an error has occurred that blocks the shipper from being registered. Registration can be a lengthy process so the caller should take into account that the thread will be blocked. (registration can take up to 5 minutes with UPS) Parameters: An enumerated constant that represents the carrier that the shipper is to be registered with. A pointer to an IMap interface that contains that parameters that are required to register the shipper.
GetManifestParameters()	This method will be used to get a collection of all parameters that are required to send a manifest to a particular carrier. The caller will then fill the collection with the required parameters for the particular carrier and use the SendfManifest method to send the electronic manifest. Parameters: An enumerated constant that represents that shipper that the electronic manifest will be sent to. A pointer to an IMap interface that will be filled in with the parameters that are required to send the electronic manifest.
SendManifest()	This method will be used to upload an electronic manifest to a specific carrier. This method will be called asynchronously and will return right away. The manifest will be queued for sending and a connection to the carrier will be initiated. Parameters: An enumerated constant that represents the carrier that the electronic manifest should be sent to. An IMap interface that contains the parameters that are necessary

	to send the electronic manifest. A string of the UNC filename for the electronic manifest.
Properties	
None are required.	

Table 9.3.6-2. conn_ICarrierConnect Interface Description.

Interface conn_IDirectTrack

This the interface that will be used for tracking directly with the carrier.

Interface conn_IDirectTrack	
Identifier	
ProgID	MSI.conn_IDirectTrack
Initialization/Un-initialization	
Connect()	This method will establish a connection to the carrier.
Disconnect()	This method will disconnect from the carrier.
Methods	
GetPackageTrackingParameters()	This method will be used to get a collection of parameters that are required to track a package. Parameters: An IMap interface that will be filled in with the parameters that are required to track the package.
TrackPackage()	The method will do the direct tracking of the package with the carrier. Parameters: An IMap interface that contains that parameters that are required to track the package. A pointer to an IStream interface that will be filled in with the raw tracking data returned from the carrier.
Properties	
None are required.	

Table 9.3.6-3. conn_IDirectTrack Interface Description.

Interface conn_IWebTrack

This the interface that will be used for tracking using a carrier's web site.

Interface: conn_IWebTrack	
Identifier	
ProgID	MSI.conn_IWebTrack
Initialization/Un-initialization	
Connect()	This method is used to initial the TCP/IP connection to the carrier.
Disconnect()	This method will free up any resource that are being used for the TCP/IP connect.
Methods	
GetPackageTrackingParameters()	This method will be used to get a collection of parameters that are required to track a package. Parameters: An enumerated constant that represents that carrier that the package will be tracked through.
TrackPackage()	This is the method that will hit a carrier's WEB site to track a package. Parameters: An enumerated constant that represents the carrier that the package will be tracked through. An IMap interface that contains all of the parameters that are required to track a package. An IStream interface that will be filled with the raw HTML returned from the carrier WEB site when the package is tracked.
Properties	
None are required.	

Table 9.3.6-4. conn_IWebTrack Interface Description.

9.3.6.3.4 Interaction Diagrams

To better understand the interactions with Carrier Connectivity and other package and components of project Wolverine, interaction diagrams are presented. Each diagram depict a scenario showing one or more components interacting. The diagram do not depict what is going on inside of the Carrier Connectivity. It is treated here as a black box.

Typical interaction between a tracking client and carrier connectivity.

The following interaction diagram shows how a tracking client would use Carrier Connectivity to track a package using the carrier WEB site.

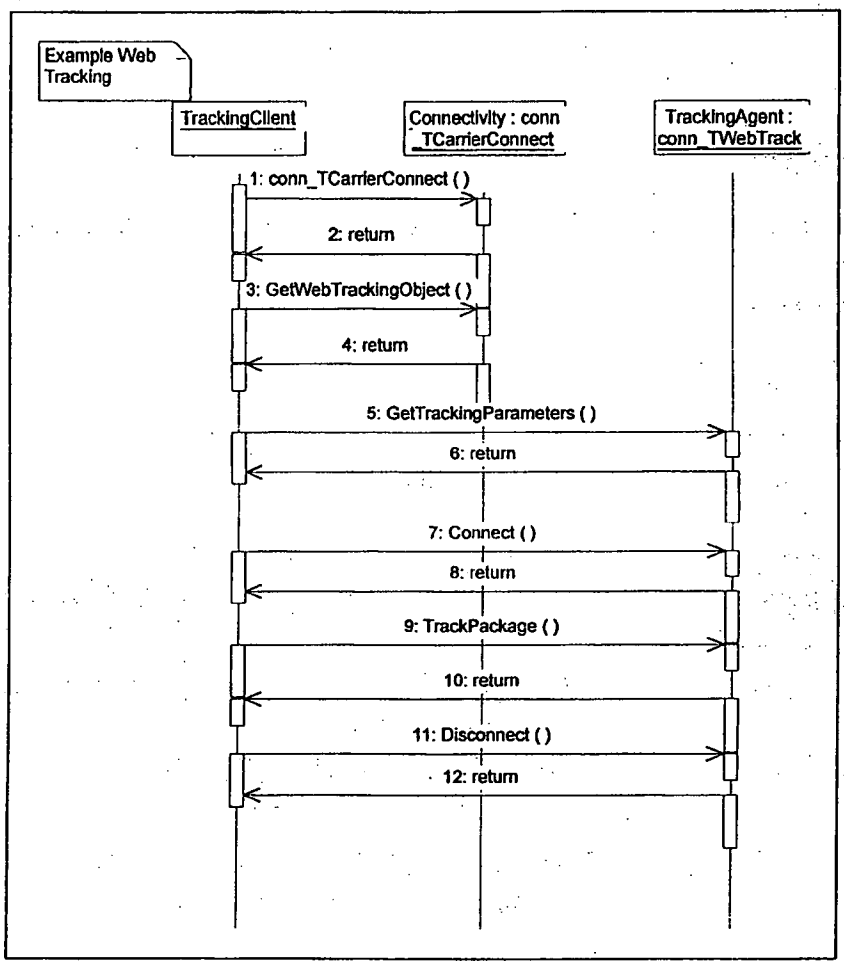


Figure 9.3.6-1. Tracking client and carrier connectivity interaction.

9.3.6.3.5 Messages

The table below defines messages that either <Gadget> Services publishes or subscribes to. For more information on messages, see the Event Services design section in this document.

Name	Description
_STATUS.INFO	
None are required.	
_STATUS.WARNING	
None are required.	
_STATUS.ERROR	
None are required.	
_JOB	
None are required.	

Table 9.3.6-5. Publish and Subscribe Messages Supported by Carrier Connectivity Services.

9.3.6.4 Implementation Details

9.3.6.4.1 Classes

Class conn_TCarrierConnect

This is the class that implements that conn_ICarrierConnect Interface.

class: conn_TCarrierConnect	
Constructors/Destructors	
~conn_TCarrierConnect()	When the constructor is called any active connections to carriers will be disconnected.
conn_TCarrierConnect()	There will only be a default constructor.
Methods	
GetDirectTrackingObject()	This method will provide an object pointer that can used for package tracking through a direct connection to the shipper. Parameters: An enumerated constant that represents that carrier that the package should be tracked through. A pointer to an conn_TDirectTrack pointer.

GetWebTrackingObject()	This method will provide an Interface that can be used for package tracking through the Web sight of the carrier. Parameters: A pointer to a conn_TWebTrack pointer.
GetRegistrationParameters()	This method will be used to get a collect of all parameters that are required to register a shipper with a give carrier. The caller will then be able to enumerate over the collection and fill in all of the data that is required by the carrier to register a new shipper. Parameters: An enumerated constant that represents the carrier that the shipper will be registered with. A pointer to an IMap interface that will be filled in with the parameters that are required to register the shipper.
RegisterShipper()	This method will be used to register a shipper with a given carrier. This function is synchronous and will return until the shipper has either been registered successfully or an error has occurred that blocks the shipper from being registered. Registration can be a lengthy process so the caller should take into account that the thread will be blocked. (registration can take up to 5 minutes with UPS) Parameters: An enumerated constant that represents the carrier that the shipper is to be registered with. A pointer to an IMap interface that contains that are parameters that are required to register the shipper.
GetManifestParameters()	This method will be used to get a collection of all parameters that are required to send a manifest to a particular carrier. The caller will then fill the collection with the required parameters for the particular carrier and use the SendfManifest method to send the electronic manifest. Parameters:

	An enumerated constant that represents that shipper that the electronic manifest will be sent to. A pointer to an IMap interface that will be filled in with the parameters that are required to send the electronic manifest.
SendManifest()	This method will be used to upload an electronic manifest to a specific carrier. This method will be called asynchronously and will return right away. The manifest will be queued for sending and a connection to the carrier will be initiated. Parameters: An enumerated constant that represents the carrier that the electronic manifest should be sent to. An IMap interface that contains the parameters that are necessary to send the electronic manifest. A string of the UNC filename for the electronic manifest.
Properties	
None are required.	
Operators	
None are required.	

Table 9.3.6-6. conn_TCarrierConnect object Description.

Interface conn_TDirectTrackXXXX

This is an example class for direct tracking. There will be a separate class for each carrier since direct tracking with a carrier will be very different for each carrier. For example UPS used a proprietary protocol UPSLink/z that is implemented by UPSLINK.DLL provided by ups. For this reason there will be a separate class that will implement the conn_IDirectTrack interface for each carrier.

Interface: conn_TDirectTrackXXXX	
Constructors/Destructors	
conn_TDirectTrackXXXX	I do not anticipate the constructor will take any parameters.
~conn_TDirectTrackXXXX	If there is an active connection to the carrier the connect will be shut down when the destructor is callect.
Methods	
Connect()	This method will force a connection to the carrier.
GetPackageTrackingParameters()	This method will be used to get a collection of parameters that are required to track a package. Parameters: An IMap interface that will be filled in with the parameters that are required to track the package.
TrackPackage()	The method will do the direct tracking of the package with the carrier. Parameters: An IMap interface that contains that parameters that are required to track the package. A pointer to an IStream interface that will be filled in with the raw tracking data returned from the carrier.
Disconnect()	This method will disconnect from the carrier.
Properties	
None are required.	
Operators	
None are required.	

Table 9.3.6-7. conn_TDirectTrackXXXX Interface Description.

Interface conn_TWebTrack

This is the class that will implement the conn_IWebTrack interface. Unlike the direct track class there will only be a single class that will implement the WEB tracking interface. The reason that there will only be one class to implement the WEB tracking interface is that WEB tracking is basically the same for all carriers. The only difference between web tracking for the carriers will be the URL that is used to track and the required parameters, and those can easily be data driven and be implemented by the same class.

Interface: conn_TWebTrack	
Constructor/Destructor	
conn_TWebTrack()	I do not anticipate the constructor will take any parameters.
~conn_TWebTrack()	Any TCP/IP connect will be cleaned up in the destructor.
Methods	
Connect()	This method is used to initial the TCP/IP connection to the carrier.
Disconnect()	This method will free up any resource that are being used for the TCP/IP connect.
GetPackageTrackingParameters()	This method will be used to get a collection of parameters that are required to track a package. Parameters: An enumerated constant that represents that carrier that the package will be tracked through. An IMap interface that will be filled in with the parameters that are required to track the package
TrackPackage()	This is the method that will hit a carrier's WEB site to track a package. Parameters: An enumerated constant that represents the carrier that the package will be tracked through. An IMap interface that contains all of the parameters that are required to track a package. An IStream interface that will be filled with th raw HTML returned from the carrier WEB site when the package is tracked.

Properties	
None are required.	
Operators	
None are required.	

Table 9.3.6-8. conn_IWebTrack Interface Description.

9.3.6.4.2 Class Diagrams

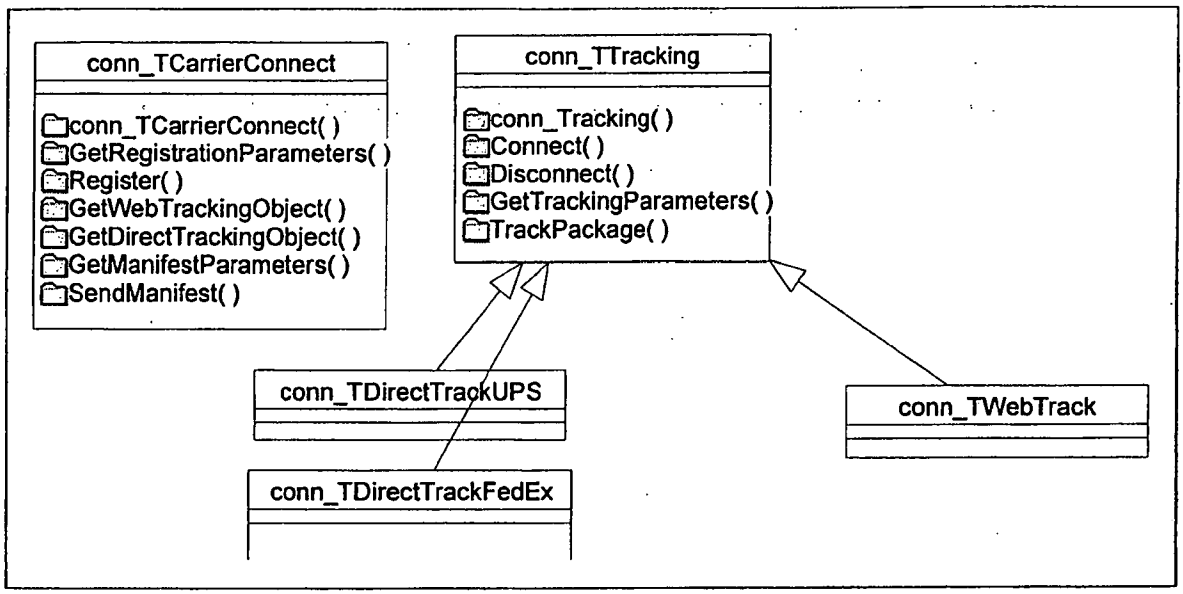


Figure 9.3.6-2. Class Hierarchy Diagram for Carrier Connectivity..

9.3.6.4.3 Free Function

None are required.

9.3.6.4.4 Interaction Diagrams

None are required.

9.3.6.5 External Dependencies

The table below lists dependencies that this package has on other components and database tables.

Type	Name	Description
COM Object	IDictionary	Microsoft dictionary
DLL	MFC42.DLL	Microsoft MFC support DLL
Static Library	ATL	Active X Template Library >
DLL	UPSLINK.DLL	UPS communications DLL.

Table 9.3.6-9. Carrier Connectivity Services External Dependencies

9.3.6.6 Requirements Traceability Matrix

The table below traces requirements from the Server functional requirements to components of

the Carrier Connectivity service.

Requirement ID	Class/Function/Interface	Comments

Table 9.3.6-10. Carrier Connectivity Services Requirements Traceability Matrix

EXHIBIT “J”

From: Jinyue Liu [IMCEAEX-_O=VELLEB+2C+20INC+2E_OU=VELLEB_CN=RECIPIENTS_CN=JINYUE@iship.com]
Sent: Thursday, January 22, 1998 2:33 PM
To: MoveIt! Devel
Subject: Notes on optional tag of enum in IDL

Place the optional tag name of enum defined in an IDL in front of the definition {...} so that this tag name will appear in the TLH file when you #import the TLB generated by this IDL.

for example,
typedef [v1_enum] enum EConnectType {dbs_kShipping, dbs_kTracking, dbs_kReport, dbs_kAccount } EConnectType;
when #import'ed, will generates
enum EConnectType

```
{
   dbs_kShipping = 0,
   dbs_kTracking = 1,
   dbs_kReport = 2,
   dbs_kAccount = 3
};
```

in the .TLH file.

But
typedef [v1_enum] enum {dbs_kShipping, dbs_kTracking, dbs_kReport, dbs_kAccount } EConnectType;
generates

```
enum __MIDL___MIDL_itf_Database_0000_0001
{
   dbs_kShipping = 0,
   dbs_kTracking = 1,
   dbs_kReport = 2,
   dbs_kAccount = 3
};
```

Following typedef without trailing tag will not compile.
typedef [v1_enum] enum EConnectType {dbs_kShipping, dbs_kTracking, dbs_kReport, dbs_kAccount } ;

-Jinyue

From: Paul McLaughlin [IMCEAEX-_O=VELLEB+2C+20INC+2E_OU=VELLEB_CN=RECIPIENTS_CN=PAULM@iship.com]
Sent: Tuesday, March 24, 1998 8:06 PM
To: MoveIt! Devel
Subject: Shipping Station Setup


Shipping Station Setup.doc

You now have two choices to when you want to use a shipping station:

1. Choose either the shipping station in the main conference room or the one in William's old office on the device rack on the bottom shelf on the left.

2. Put yourself through a little hell and setup your own PC as a shipping station and print labels on the shipping station in William's old office remotely. You don't need a scale.

You will have to follow this document to get option (2) to work:

<<Shipping Station Setup.doc>>

With either option, you should logon to NT using the following user information:
USER: sstation
Password: hamsandwich
Domain: moveit!

-LowRider

To setup a shipping station, do the following:

UPDATE NT:

- You must be running NT 4.0 with SP3.

UPDATE IE:

1. The Shipping Station requires IE 4.0.
2. Install IE 4.01 for NT. (This is on \\macarthur\scratch).
3. Close IE.
4. Logoff NT.
5. Logon NT as "**sstation**" with password "**hamsandwich**".
6. After IE gets done updating your system, right-click IE on the desktop.
7. Select Properties.
8. Choose the Security dialog tab.
9. Select "Local intranet zone"
10. Click the "Low" radio button.
11. Choose the Advanced dialog tab
    Under the "Browsing" category
    Turn off, "Show welcome message each time I log on"
    Turn on, "Launch browser in full screen window"
12. Press OK to close the Properties dialog.

UPDATE CLIENT FILES:

1. Copy all the files from "N:\PAULM\SHIPPINGSTATION" to a local directory of your choice (using your Windows directory is an example).
2. "CD" into the directory you chose in the previous step, and type "SSSetup".

UPDATE MONITOR RESOLUTION:

- Set your screen resolution to 800x600, if you want the end-user viewing experience.

CONFIGURE PRINTERS:

- Setup a network printer driver to print to "`\\QA2\Eltron 2044`" by using the Add Printer wizard in Start/Settings/Printers. You may have to type in the net address of the printer to get the connection.

MISCELLANEOUS FINAL IE SETTINGS:

1. Launch IE.
2. The shipping station URL is "http://gambit/ShippingNetwork/ShippingStation". Either make this URL your IE startup page, or save a link to it in your Favorites. Make sure that the URL doesn't point to the Setup subdirectory of ShippingStation if you save it as your home page.
3. When IE launches, there may be an IE toolbar on the top. If so, right-click the toolbar and choose "Autohide".
4. If you chose 800x600 resolution, turn on "Autohide" in Start/Settings/Taskbar. IE should be truly "full screen" after doing so.

SETUP THE SHIPPING STATION APPLICATION

1. Tab to "Drop Off ID" and type "MoveIt! Headquarters".
2. Tab to "Password" and type "Battlezone" (yes, case sensitive).
3. Tab to "Shipping Station Nam " and type whatever your heart desires.
4. Click the Submit button.
5. The Setup Control page doesn't work right now, so after it loads, press the "Goto Clerk Logon" button.
6. Logon as your email name for the Clerk ID (e.g. I logon as "paulm") and your password is initially "password".
7. Click Configuration
8. Click Printers
9. Choose "Eltron 2044" for the Label and Receipt printers.
10. Select the driver attached to "`\\qa2\eltron 2044`" for the "local driver" settings for the label and receipt printers.
11. You may leave the Report printer settings "not configured" for now as EOD isn't implemented yet.
12. Choose "Manual Override" from the list of Supported Scales. You may leave "Local Port" as is.
13. Click the "Submit New Configuration" button.
14. Click the Logoff button and you're ready to ship packages!

Come see me if you have any problems.

-Paul McLaughlin

Fr m: William Smith [IMCEAEX-_O=VELLEB+2C+20INC+2E_OU=VELLEB_CN=RECIPIENTS_CN=WILLIAM@iship.com]
Sent: Saturday, September 19, 1998 1:35 PM
To: Shaindell Goldhaber; David Bennett; Chad Mentzer
Cc: John Dietz
Subject: RE: Proposed menus

An implementation for voiding a package if you are not logged in:
An edit field where you type the iShip tracking number
press a button and it retrieves the least amount of info. for the
user to verify that the package is the one they shipped. At this
point you can void it or reprint the label.

Problems I see:
People messing with are system and trying to guess
tracking numbers and voiding packages or reprinting
another person's labels.

The other implementation is to have the person send us
an email if they want to void a package or reprint a label.
We send them an email that gives them a special number
they enter that only works with that package. We can
check the sender's e-mail against the e-mail address of
the person who shipped the package.

W.

> -----Original Message-----
> From: Shaindell Goldhaber
> Sent: Friday, September 18, 1998 5:26 PM
> To: William Smith; David Bennett; Chad Mentzer
> Subject: Proposed menus
>
> David doesn't like the idea of including left-hand menu buttons for section start pages. However, I've included them here for the sake of discussion. Also, I've added Reprint Label and Void a Package to the shipping menus. I'm not sure what the implementation would look like for users who are not logged on, however.
>
> --Shaindell
>
>
> Logged Off
>
> Home
> Log on
> Welcome
> Sign up
> Help
>
> Shipping
> Log on
> Ship a Package
> Reprint Label (?)
> Void Package (?)
> Compare Services
> Drop off Locator
> Help
>
> Tracking
> Log on

> Track a Package
> File a Claim
> Help
>
> Support
> Log On
> Contact Us
> Feedback
> How To
> Help
>
> Logged On
> Home
> Log off
> Welcome
> Address Book
> Preferences> ...>
> Help
>
> Shipping
> Log off
> Ship a Package
> Reprint a Label
> Void a Package
> Compare Services
> Drop off Locator
> Help
>
> Tracking
> Log off
> Track a Package
> View Shipping Log
> File a Claim
> Help
>
> Support
> Log Off
> Contact Us
> Feedback
> How To
> Help
>
>

From: William Smith [IMCEAEX-_O=VELLEB+2C+20INC+2E_OU=VELLEB_CN=RECIPIENTS_CN=WILLIAM@iship.com]
Sent: Wednesday, September 09, 1998 5:44 PM
To: Chad Mentzer
Cc: The Teamsters; The Raiders; The Wolverines
Subject: Configuration Interface Design.

This design can be reused for all objects that need configurations
tied to an OID. Use Gary's config services otherwise.
Paul M - I believe the CONFIGURATION database table is
not being used. It should be removed in favor of either using
Gary's config items or a special configuration tabel modeled
after what Chad is doing.

CHAD:

Have something like this returned for a Configure() method on the acct_IAccount and acct_IUser interface.
There would not be a COCLASS exposed. It is modeled after dbs_IConfigServices. Steal the code.
Default values for new accounts at time of registration should reside in Gary's config services.
Don't store config. values that need have database rules enforced
(for example cascade delete). Add columns to your account and accountuser db tables
in these cases.

Add this method to acct_IUser & acct_IAccount

```
HRESULT Configuration([out, retval] acct_IConfiguration** ppIConfiguration);
```

Define this interface in your IDL

acct_IConfiguration

```
        object,
        uuid(3CAE1318-A17A-11D1-9636-00A02475D4D9),
        dual,
        helpstring("acct_IConfiguration Interface"),
        pointer_default(unique)

interface acct_IConfiguration : IDispatch

        [
                propget,
                id(1),
                helpstring("property Item")

        ]
        HRESULT Item([in] BSTR Scope, [in] BSTR ItemName, [in, optional] VARIANT GetDefaultPutDescription,
[out, retval] VARIANT *pVal);
        [
                propput,
                id(1),
                helpstring("property Item")

        ]
        HRESULT Item([in] BSTR Scope, [in] BSTR ItemName, [in, optional] VARIANT GetDefaultPutDescription,
[in] VARIANT newVal);

                id(2),
                helpstring("method Delete")
```

```
        HRESULT Delete([in] BSTR Scope, [in] BSTR ItemName);

            id(3),
            helpstring("method Exist")

        HRESULT Exist([in] BSTR Scope, [in] BSTR ItemName, [out, retval] int *pExists);

            id(4),
            helpstring("method Items")
        ]
        HRESULT Items([in] BSTR Scope, [in] util_IMap *ItemMap);
};
```

The Items() method takes a map filled with keys (itemnames) that you want retrieved. After the call, the key values are filled in. This is for speed of getting multiple items (preferences).

Scope would be for example:
"DOCUMENTS"
Item
"ShippingLabelDevice"
Value
"\\macarthur\hp5msix"
Description (optional)
"Default label printer"

TWO(2) database tables need to be created.

ACCOUNTCONFIG
and
ACCOUNTUSERCONFIG

These should have the
sam design as the CONFIGITEMS table with the exeption that
keys have to be added (unique key is the composite of the asteriks items):
For ACCOUNTCONFIG

ACCOUNTNO	(VARCHAR 6)
SCOPE	(VARCHAR 38)
ITEM	(VARCHAR 255)
VALUE	(VARCHAR 255)
DESCRIPTION	(VARCHAR 255)

For ACCOUNTUSERCONFIG

USERID	(VARCHAR 35)
ACCOUNTNO	(VARCHAR 6)
SCOPE	(VARCHAR 38)
ITEM	(VARCHAR 255)
VALUE	(VARCHAR 255)
DESCRIPTION	(VARCHAR 255)

Cascade delete should be enforced if the account or accountuser is deleted as appropriate. Having configuration items is OPTIONAL.

William Smith
V.P. Engineering
iShip.com, Inc - The Internet Package Shipper

THIS PAGE BLANK (USPTO)

From: Paul McLaughlin [IMCEAEX-_O=VELLEB+2C+20INC+2E_OU=VELLEB_CN=RECIPIENTS_CN=PAULM@iship.com]
Sent: Monday, November 30, 1998 3:20 PM
To: William Smith; Chad Mentzer; Talal Karkutly
Subject: RE: Columns to add to the portal table.

I need to know exactly, given a SiteOID, how to get the associated Carrier Account.

Currently, I go through the following path in many of my stored procedures to resolve a SiteOID to a CarrierAccount:

SiteAndCarrier -> Account -> AccountAndCarrierAcnt -> CarrierAccount

This works, because the SiteOID is only in the Account table once when the AccountTypeName is either "SiteAccount" or "ParentAccount". After the removal of SiteOID from Account I do not see how I can accomplish the Carrier Account resolution. I can't use AccountAndSite, because that table does not resolve to one Account.

It is almost like we need a SiteAndCarrierAccount table?

-LowRider

> -----Original Message-----
> From: William Smith
> Sent: Monday, November 30, 1998 11:49 AM
> To: Paul McLaughlin; Chad Mentzer; Talal Karkutly
> Subject: RE: Columns to add to the portal table.
>
> Yes, I know that was its original intent,
> but w/o it we can't share accounts across sites.
> Comments?
>
> W.
>
> -----Original Message-----
> From: Paul McLaughlin
> Sent: Monday, November 30, 1998 11:47 AM
> To: William Smith; Chad Mentzer; Talal Karkutly
> Subject: RE: Columns to add to the portal table.
>
> Remember, this was supposed to an exclusion table. I don't see how it can fulfill the role.
>
> -LowRider
>
> -----Original Message-----
> From: William Smith
> Sent: Monday, November 30, 1998 11:40 AM
> To: Paul McLaughlin; Chad Mentzer; Talal Karkutly
> Subject: RE: Columns to add to the portal table.
>
> There is a table called ACCOUNTANDSITE that should
> fulfill this role.
>
> W.
>
> -----Original Message-----
> From: Paul McLaughlin
> Sent: Monday, November 30, 1998 11:34 AM
> To: Chad Mentzer; William Smith; Talal Karkutly
> Subject: RE: Columns to add t the portal table.
>

> If SiteOID is removed from ACCOUNT, then there is no way to d termine what carrier accounts belong to a site.
>
> Removing SiteOID from ACCOUNT is going to break EOD, the DOL, and probably rating.
>
> -LowRider
>
> -----Original Message-----
> From: Chad Mentzer
> Sent: Monday, November 30, 1998 11:20 AM
> To: William Smith; Talal Karkutly
> Cc: Paul McLaughlin
> Subject: RE: Columns to add to the portal table.
>
> You are correct. Both the SiteOID and Announcements Flag should be removed.
>
> ---
> Chad Mentzer
> iShip.com
> The Internet Package Shipper> (tm)>
> Phone: (425) 602-5032
> Fax: (425) 602-5025
> e-mail: chad@iship.com
>
> compare shipping with iShip.com at http://www.business.msn.com
>
> -----Original Message-----
> From: William Smith
> Sent: Saturday, November 28, 1998 12:54 PM
> To: Talal Karkutly
> Cc: Paul McLaughlin; Chad Mentzer
> Subject: Columns to add to the portal table.
>
> PARENTNO varchar(6) - The parent account number for all new user or site accounts added through this portal.
> ACCOUNTPREFIX varchar(3) - The prefix to append before all accounts created via this portal
> DOMAIN varchar(35) - The security domain under which all new users should be created and logged in under.
>
> In looking over the ACCOUNT table, it looks like
> SiteOID
> should be removed. An account doesn't have a site. If we want to have
> an account template for new account users that may have a default site,
> then that should be handled using a special accountuser record to act as a
> template. Chad/PaulM any comments?
>
> In the ACCOUNTUSER table, the announcements flag should be removed since it
> now exists in the ACCOUNTUSERCONFIG table. Chad/PaulM any comments?
>
>
>
> William Smith
> V.P. Engineering
> iShip.com, Inc - The Internet Package Shipper
>
>
>

From: ishIp [IMCEAEX-_O=VELLEB+2C+20INC+2E_OU=VELLEB_CN=RECIPIENTS_CN=ISHIP@iship.com]
Sent: Tuesday, December 01, 1998 6:44 PM
To: William Smith
Subject: Your Shipping Request # M AZHDY6 DYF2C0

At 3:44 PM on Tuesday, December 1, 1998 you shipped a package with iShip Package Number M AZHDY6 DYF2C0 via Next Day Air Saver. to Carl Lewis located in SEATTLE, WA.

You must drop off your package before 5:00 PM on Tuesday, December 1, 1998 in order for your package to arrive at its destination by Wednesday, December 2, 1998.

Your selected Drop-Off location is:

UPS Drop Box
bellevue, WA 98005

Your package weight is 10.00 lbs.
The billable weight is 10.00 lbs.
The charge for this package is:

Base Service Charge:
$18.75

Total:
$18.75

To check on the Status of the Package, go to:

http://TESTNOC/ShippingNetwork/trk.asp?T=MAZHDY6DYF2C0

To view any confidential information about the Request/Package you must, however, go to http://TESTNOC and Log On to your account.

Thank you for shipping with iShip.com!

Fr m: Paul McLaughlin [IMCEAEX-_O=VELLEB+2C+20INC+2E_OU=VELLEB_CN=RECIPIENTS_CN=PAULM@iship.com]
Sent: Tuesday, July 20, 1999 5:20 PM
T : iShip Devel
Subject: Some very interesting ActiveX/IIS issues have been solved

Ordinarily, this might have been a day or two road block, but since Mark, Reichie and I pulled together for about a half an hour, {voice:select("Hanz and Franz")} "We crushed those girly man bugs like the puny insects they are. Ya!" {voice:select ("default")} iShip.com rocks!

Problem 1: ActiveX controls refused to download even though client system was clean.
Solution: The virtual directory "bin" (that points to your equivalent of $/WebApps/bin), had in IIS the "Execute" permission set for the directory, when it should have been only "Script".

Problem 2: CSZ Database ActiveX control installed correctly, but CMS still thought it had not.
Solution: As far as COM was concerned, it did install correctly. All the files where registered and on the client's system. However, the Z5PLUS.TXT needs to be in the same directory as the MSI_ShippingStation.dll and my JScript calls a COM function in the dll to check this. It was this call that was failing for some reason.

So why then was the ActiveX control installing correctly for COM, but the Z5PLUS.TXT file was not in the "downloaded program files" directory when viewed from a command prompt? Well, turns out that when the ActiveX Component Download process checks to see if a file already exists, it doesn't just check the target directory of the file (specified in the .INF file), but rather it does the standard "search for file" process as defined by the Win32 API function SearchPath(). This algorithm searches several different locations, the last of which is the PATH, and guess what file was coincidentally found along the PATH? Yep, Z5PLUS.TXT was in the "...\debug\bin" directory which was in the path (as it is for most of us developers). So, we nuked the Z5PLUS.TXT file from "debug\bin", tried the CMS install again and everything worked fine.

I am going to check to see if my use of the default destination directory causes the search to happen, or if I tell it explicitly to use the "downloaded program files" directory that it will not search for the file in the aforementioned manner. I'll let you know.

Learn something new every day.

Paul R. McLaughlin
iShip.com
Your Internet Package Shipper (tm)
Senior Software Developer
(425)602-5046 (work)
(425)602-5025 (fax)
(425)556-9257 (home)
(425)444-9257 (cell)
e-mail: paulm@iship.com
http://www.iship.com

compare shipping with iShip.com at http://www.business.msn.com/shipping/compare.asp

From: Paul McLaughlin [IMCEAEX-_O=VELLEB+2C+20INC+2E_OU=VELLEB_CN=RECIPIENTS_CN=PAULM@iship.com]
Sent: Wednesday, July 21, 1999 2:57 PM
To: William Smith; iShip Devel
Cc: Ken Sinclair
Subject: RE: Caching Data On The CMS Client:

Importance: High

Here is how it will be done. I will use the SiteInformation dialog as an example throughout:

Overview:
1. The developer in charge of data that needs to be cached on the client, will develop a global JScript object exposing the data as properties and will be contained in the cmsApps.asp client-side script.
2. UI that presents this data, for example the Site Information dialog, will continue to use the tx file they have to get the data, however, the tx file will update the global object (via the object's exposed Update() method) and then the dialog will instead update its UI with the property values from the object.
3. Client-side script that cares about such data will always get it from the global objects.
4. I will write a server and client-side "command processor" that continuously polls our servers for new commands to run, one of which can be "submit a tx file". This will be the exact same tx file that the dialogs use to get the latest data. The client-side command interpreter will then also call the global object's Update() method exactly as in item (2).

Note that the development of the global objects and their integration with the dialogs are not effected by my development of the command interpreters. The dialogs and global objects should be completely unaware of any existence of my command interpreters.

TODO: Detailed Development Tasks
Stuff you need to do:
1a. If you have data that needs to be cached, then you need to create a new ".js" file that contains an object named after your UI component (like William recommends below). The .js file should also be named after the object name.
1b. The JScript object will expose a property for each data element that is cached. For example, gSiteInformation.City and gSiteInformation.State would return the respective data items.
1c. The JScript object will support a member function called Update() which takes the exact same object that is returned by the tx file when it calls its equivalent of "parent.txComplete(txResult)".
1d. All JScript functions in the .JS file will be prefixed with a unique prefix that is named after the object. This is necessary because many .js files will be included in a single page and we must avoid name conflicts. For example, the JScript function for the object's Update function property might be prototyped as "function si_Update(txResult)" and then of course later on it would be assigned to the Update property in the constructor of the SiteInformation object as "this.Update = si_Update;".

2a. cmsApps.asp will be the container for all the cached data.
2b. cmsApps.asp will be modified with "<script language="jscript" src="yourobject.js"></script>" for each object that is needed.

3a. Your UI components current have some sort of "txGet....asp" to retrieve the data. This file will stay the same with one exception: you must rename the "parent.txComplete(txResult);" line of script to be something unique named after your data set. For example: "parent.txCompleteSiteInformation(txResult);"
3b. Move and modify the code of "txCompleteSiteInformation(txResult)" that updates the UI from the txResult object, to a separate function. In that function, instead of getting your data from the txResult object, you must get the data from the properties of the global cached object.
3b. Modify the code of "txCompleteSiteInformation(txResult)" from updating the UI directly, to merely passing the exact same txResult object returned from your tx file, to the cached object's Update() function.

4. All client side script that cares about this cached data, needs to get the data from the appropriate global objects.

Stuff I need to do: (none of which effects your development, but just FYI)
1. Complete command table design and add schema to database.
2. Add data to command tables to facilitate running tx files.
3. Write server-side command processor tx file.

4. Write client-side command interpreter.

NOTE: Until my command interpreter thing works, the object will NOT CONTAIN any data until you open and close the corresponding UI component. In other words, you must open and close the Site Information dialog to "refresh" the data in the SiteInformation global object.

EXAMPLES:

SiteInformation.js:
```
function si_Update(txResult)
{
   // update all data properties with returned tx data.
  this.City = ...["city"];
}

function si_InitProps()

  this.City = "";


function gSiteInformation()

  this.Update = si_Update;
  si_InitProps();


gSiteInformation = new gSiteInformation();
```

cmsApps.asp:
```
...
<script language="JScript" src="/cms/lib/SiteInformation.js"></script>
```

txGetSiteInformation.asp:
```
...
parent.txCompleteSiteInformation(txResult);
```

The SiteInformation dialog:
```
...
function UpdateUI(oSI)
{
  document.frmSI.editCity.value = oSI.City;  // substitute with your objects of course.
  ...


function txCompleteSiteInformation(txResult);
{
  var oSI = dialogArguments.cmsAppsWndPtr.gSiteInformation;  // or however you pass it into your dialog.
  oSI.Update(txResult);
  UpdateUI(oSI);
```

hope this helps. Don't hesitate to ask any questions or send comments my way. They will be appropriately filed. ;-)

-LowRider

> -----Original Message-----
> From: William Smith
> Sent: Tuesday, July 20, 1999 8:58 AM
> To: iShip Devel
> Cc: Ken Sinclair
> Subject: Caching Data On The CMS Client:
>
> This is HIGH priority.
>

> If you worked on the Preferences, Center Information or Scales and Printers dialogs in
> CMS, you must ASAP provide a JavaScript object that exposes the current settings to the
> client side code. Shipping needs access to this information. There is a CMS app area
> that can be used to cache this data globally. Obviously it must remain in sync with changes
> made via the dialogs. One way is to have properties on this object that expose the current values
> and methods that do the transactions. Please send out to the development with sample code,
> how to access the current values on the client side. I would anticipate that a Preferences,
> ScalesPrinters and a SiteInformation object or some other name would exist.
>
> Notice I used SiteInformation NOT CenterInformation. Please think about the fact
> that the code you are creating today will be used in places other than MBE.
>
> Sean Hu has done this very well by creating a JavaScript object for the Hardware settings.
> Sean please work with Reichie and MarkB (if they are the right ones).
>
> PaulM - The COMMAND function you are working on, should have a way to
> refresh these objects on the client side if a change is made on the backend
> using the Administration system.
>
>
> William Smith
> V.P. Engineering
> iShip.com, Inc - Your Internet Package Shipper
>
>
>

From: Jinyue Liu [IMCEAEX-_O=VELLEB+2C+20INC+2E_OU=VELLEB_CN=RECIPIENTS_CN=JINYUE@iship.com]
Sent: Monday, July 26, 1999 2:50 PM
To: Glenn Crowe
Cc: William Smith
Subject: RE: MBE Customer ID Requirement.

The input will be custom id (an integer) and the output will be the check digit.

```
function Mod10(int CustomerID)
{
        int Odd  = 0;
        int Even = 0;
        int i = 0;
        while (CustomerID > 0)
        {
                int Val = CustomerID % 10;

                CustomerID = CustomerID / 10;

                // Separately add up the number in the odd and
                // even positions of the input string.

                if ((i+1)%2)
                        Odd  += Val;
                else
                        Even += Val;

                i = i + 1
        }

        // Add the sum of the odd position characters
        // to twice the sum of the even position characters.
        // Then get the remainder of this sum.

        int CheckDigit = (Odd + (2 * Even)) % 10;

        // The check digit is 0 if the remainder is 0.
        // Otherwise, the check digit is 10 - the remainder.
        if (CheckDigit > 0)
                return '0' + (10 - CheckDigit);
        else
                return '0';
}
```

> -----Original Message----
> From: Glenn Crowe
> Sent: Friday, July 23, 1999 3:02 PM
> To: Jinyue Liu
> Subject: FW: MBE Customer ID Requirement.
>
> Jinyue,
>
> This is the algorithm that William wants me to implement.
>

> Thanks!
> ---
> Glenn Crowe
> Associate Developer
> iShip.com
> Your Internet Package Shipper (tm)
> glennc@iShip.com
> 425.602.5044
>
> Check us out at http://iShip.com.
>
>
> -----Original Message-----
> From: William Smith
> Sent: Friday, July 23, 1999 2:56 PM
> To: David Bennett
> Cc: Glenn Crowe
> Subject: MBE Customer ID Requirement.
>
> The customer id will be a 15 character field with the following format:
>
> MCMS##########
>
> The human readable format shall be:
>
> MCMS-###-####-####-#
>
> The prefix "MCMS" stands for MBE Counter Manifest System
>
> The middle 10 digits are 0 padded integer starting at 1.
> Each newly generated Customer # will be a simple increment
> to the last number assigned customer number.
> The human readable portion of this number is hyphenated just
> like a phone number that includes the area code.
>
> The final digit is a MOD 10 (UPS Algorithm) of the 10 digit number
> portion of the customer id. The Check digit does not include the
> "MCMS" prefix.
>
> In all cases where the customer id to read in the CMS, EPSO applications or
> on a report, the human readable version shall be display.
>
> In all cases where a customer id can be entered into the system, the data
> entry field shall support both the human readable version and the raw
> customer id. Data entry shall except either hyphens or spaces the
> delimit fields.
>
> Implementation notes:
> The current value of the MBE customer id will be stored in the _AccountConfig database
> table associated with the MBE master iShip Account#.
>
> The stored procedure that retrieves the next available customer id shall take an iShip Account# as
> input. This account# shall be used to locate the AccountConfig item. If this item does not exist,
> it shall be created, and a customer id with the number portion set to "1" shall be returned.
>
> The stored procedure shall return a complete customer id in non-human readable format as a string.
>
> The stored procedure must make sure the deadlocks cannot occur and the transaction is atomic.
>
> The account config item shall be named:
> MBE.CUSTOMERID.CURRENT
>
> William Smith
> V.P. Engineering

> iShip.com, Inc - Your Internet Package Shipper
>
>
>